DISPERSE: A Decentralized Architecture for Content Replication Resilient to Node Failures

Santhanakrishnan Anand, *Member, IEEE*, Ding Ding, Paolo Gasti, *Member, IEEE*, Mike O'Neal, *Member, IEEE*, Mauro Conti, *Member, IEEE*, Kiran S. Balagani

***Abstract*—This paper introduces DISPERSE, a distributed scalable architecture for delivery of content and services that provides resilience against node failure through location-independent storage and replication of content. Current content delivery networks (CDNs) have, at least to some degree, a centralized structure thus susceptible to a single point of failure. DISPERSE addresses this limitation by implementing a fully de-centralized structure. DISPERSE is a two-layer architecture: the first layer (front-end layer) exposes services (e.g., Web, SFTP) to clients; the second layer (back-end layer) provides reliable distributed storage of content and application state. Content in DISPERSE's back-end layer is stored and exchanged as Named Data Network (NDN) content objects. This allows DISPERSE to implement fine-grained, location-independent, fully decentralized content replication mechanisms.**

We validate the performance of DISPERSE under two node failure scenarios. In the first scenario, content can be stored in any DISPERSE node, and all nodes are equally likely to fail. In this scenario, we use non-linear optimization techniques to determine the optimal number of content copies under availability and latency constraints. In the second scenario, different nodes fail with different probabilities, and content is stored in nodes according to its value, node failure probability, and resource availability. This scenario is addressed as an instance of the minimum cost flow problem. Our results show that DISPERSE reduces the failure of content retrieval by five orders of magnitude compared to common CDN implementations, without significantly increasing content retrieval delay. Further, numerical results show that DISPERSE improves content availability by a factor of 1.3×-2.3× when deploying the minimum cost flow algorithm.

***Index Terms*—DISPERSE, Named Data Networking (NDN), Content Delivery Networks (CDN)**

I. Introduction

Content replication is the primary technique used to provide resilience against node failures. Content delivery networks (CDNs) [1]-[3] are one of the most popular ways of hosting and replicating content. A CDN consists of an "origin server" that hosts content uploaded in the network, and many "CDN servers" that host copies of content available at the origin server. When a client requests content, the CDN redirects the request to the geographically nearest CDN server. If the CDN server is unavailable, the client re-issues the same request to the origin server [4]-[6]. Although CDNs have now evolved to be effective at delivering content with high reliability and low latency, they suffer a fundamental limitation: if the origin server fails (e.g., due to natural hazards [7] or national censorships [8]), the performance of the entire CDN is greatly reduced, making the origin server *a single point of failure*.

In this paper, we introduce DISPERSE, a two-layer content distribution architecture (see Figure 1), which is functionally equivalent to CDNs. However, in contrast with CDNs, DISPERSE has no single point of failure. DISPERSE leverages Name Data Network (NDN) [9] to provide resilience to common node failure scenarios through scalable, efficient, and transparent replication of content and services across geographically distributed endpoints. Each DISPERSE layer is composed of a multitude of geographically-distributed hosts. By design, nodes in each layer are functionally equivalent, i.e., they can in principle serve the same requests. The front-end layer provides lightweight implementations of common services (e.g., Web, databases, SFTP). Each host in this layer stores no content, and is intended to maintain only transient states (i.e., states only corresponding to each transaction). As a result, front-end nodes act similarly to proxy servers to provide standard interfaces that clients use to access DISPERSE. This allows any node in the front-end layer to seamlessly take over the workload of another failed front-end layer node. The back-end layer implements reliable, scalable and low-latency distributed storage. This layer leverages Named Data Network (NDN) [9] for data representation, and for communication between back-end layer nodes, and between front-end and back-end layer nodes.

Contributions. The key contributions in this paper are: (1) the design of DISPERSE, a distributed and scalable architecture for content replication resilient to node failures; and (2) a detailed analysis that demonstrates the benefits of DISPERSE in comparison with CDNs under two scenarios, which we refer to as *randomized* and *targeted* node failure. While the proposed architecture can be used for content storage as well as content distribution, the focus of this work in on content distribution. Specifically, we evaluate the performance and resilience to node failure of DISPERSE in the context of content distribution.

By design, DISPERSE mitigates node failure by replicating both data (in the back-end layer) and services (in the front-end layer). However, replication incurs a cost. In this paper, we analyze the tradeoff between replication cost and corresponding increases in availability. We first consider a random node failure scenario, and devise a non-linear unconstrained optimization-based method to determine the optimal amount of replicas of content in DISPERSE, and extend it to include constraints on latency for content retrieval. Our results indicate that, while

Santhanakrishnan Anand, Ding Ding, Paolo Gasti, and Kiran S. Balagani are with the New York Institute of Technology (e-mail: {dding,asanthan,pgasti,kbalagan}@nyit.edu). Mike O'Neal is with the Louisiana Tech University (e-mail: mike@latech.edu). Mauro Conti is with the University of Padua (e-mail: conti@math.unipd.it).

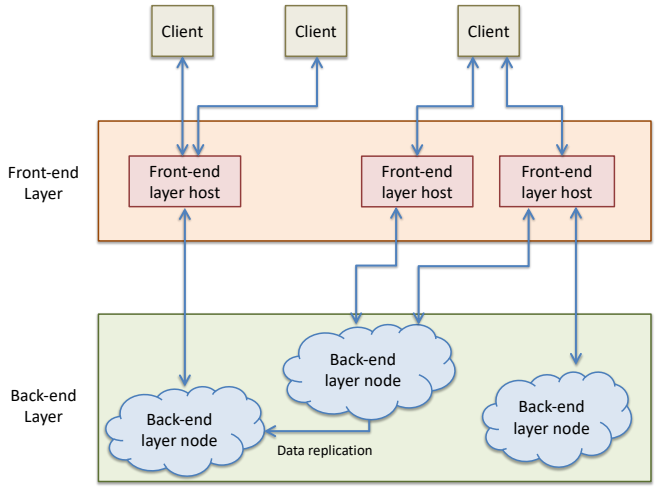


Fig. 1. Overview of DISPERSE architecture. The front-end layer acts as a proxy for the back-end layer by providing a stateless implementations of services such as Web, databases, and SFTP. The back-end layer stores copies of the content and long-term state.

there may be many factors that affect the optimal number of replicas (e.g., availability of resources, cost of storage and node failure probability), node failure probability is the most significant factor, thus motivating the need for content replication strategies that are resilient to node failure. We also compare the performance of DISPERSE and common CDN implementations, in terms of failure to retrieve content. Our results show that DISPERSE reduces the failure of content retrieval by five orders of magnitude compared to common implementation of CDNs, without significantly increasing the latency in retrieval of content.

We then develop a minimum cost flow based algorithm to optimally distribute content in a network where nodes fail with different probabilities (e.g., as a result of an attack). Further, we consider the case in which different content has different priority, and the costs incurred in storing content in different nodes are different. We develop bounds in the number of re-assignment of content to different nodes. Numerical results show that content availability increases from about 1.33× to 2.3× when deploying the minimum cost flow algorithm as against the random node failure scenario. The algorithm increases content availability for high priority content, with a trade-off of reduced availability of low priority content.

Organization. The rest of the paper is organized as follows. Section II lists the related literature on content replication. The DISPERSE architecture is described in Section III. Content replication strategies in DISPERSE under randomized and targeted node failure scenarios are discussed in sections IV and V, respectively. We conclude in Section VI.

II. Background and Related Work

Current content replication strategies can be categorized into heuristic approaches [10], game theoretic approaches [11], [12], flow control and load balancing based approaches [13], [14] and utility based approaches [15], [16], [17].

Kangasharju et al. [10] formulated the content replication problem as a constrained shortest path problem, and developed heuristics for content replication in CDNs. Khan et al. [11] present a game theoretic strategy in which users bid for resources, based on the value they assign to content. The authors discuss multiple auction models for content replication. Pollatos et al. [12] studied a Leff-Wolffe-Yu (LWF) network model wherein users store content according to a game theoretic approach to minimize cost of storage due to utilization of resources. The authors obtained pure strategy Nash equilibria and network stability conditions. La et al. [14] presented a load balancing approach for content replication under shared capacity constraints. Ganguly et al. [13] presented a cross layer architecture called SPIDER where they explored the use of optimal widest path trees in conjunction with point to point TCP flow control mechanisms. Wu and Li [15] presented a utility-based content replication strategy for video on demand in peer to peer networks. Cache-aware and scalable routing for reliable information retrieval was presented in [16]. In [17], Moharir and Karamchandani presented a fractional knapsack approach for content replication. They assigned a value to content, based on the frequency of requests and assigned weights depending on the resources consumed. Li and Simon [18] present an integer programming based push strategy for CDNs and show that it is better than caching for vide traffic. Similar integer programming approaches to minimize storage cost and latency was discussed in [19] and [20].

Most approaches in the literature only consider content replication based on resource availability, or aim at minimizing access cost. However, they do not consider failure of nodes into account. The closest work to this paper that takes online availability of peers into account is the paper of Lin et al. [21]. In this paper, the authors considered different fractions of content in different peers, and analyzed content availability of specific content when some of the peers go off-line. The main difference between our work and [21] is that in [21] node availability, content priority, and other factors are listed but replication is not optimized based on these factors. To the best of our knowledge, this is the first research work that discusses content replication optimized for content priority, node failures and resource availability at the nodes that store content.

There is a body of work (see, e.g., [22], [23], [24], [25]) introducing hybrid CDN-P2P architectures. These architectures aim to combine the scalability and robustness of P2P networks, with the reliability of CDNs. Similarly to traditional CDNs, they require the origin server to distribute and replicate content to their various nodes. As a result, the origin server represents a single point of failure, even when it is replicated in several hosts. For this reason, work on CDN-P2P does not analyze origin server failure scenarios.

A. NDN Background

Messaging and routing between DISPERSE is based on core ideas introduced by Named Data Networking (NDN) [9]. NDN is a prominent instantiation of the general content-centric networking paradigm [26], in which clients request content directly by name, rather than by the location of the host serving it (e.g., `www.cnn.com/politics`).

NDN defines two types of packets: *interest packets*, and *content objects*. Clients' requests for content are represented

by interest packets, which include the name of the content being requested, as well as metadata that helps the network to retrieve the desired content. When a router forwards an interest packet, it stores the packet's name and incoming interface ID in a data structure called Pending Interest Table (PIT). Information in the PIT is later used to forward the corresponding data packet to the consumer(s) who requested it. Because interest packets do not indicate which host is expected to serve the content, routers can forward them to either the appropriate content producer, or to a nearby cache when available. To determine how to forward interest packets, routers include a name-based Forwarding Information Base (FIB), populated using routing algorithms similar to those used with IP.

Because content objects returned to users might be hosted by any server or router on the network, trust on the content cannot be established based on the communication channel between the client and the device serving a particular content object. Instead, in NDN each content object is signed individually. This enables consumers to determine whether the content returned by the network is legitimate.

While NDN is efficient at distributing content, it requires specialized software to access it. DISPERSE addresses this issue by making NDN messaging and routing transparent to clients, which interact exclusively with front-end layer nodes.

III. DISPERSE Design

DISPERSE consists of two layers: the front-end layer, which is composed of a number of hosts, which are in charge of receiving, translating, dispatching, and responding to requests from clients (i.e., producers and consumers); and the back-end layer which stores and replicates data objects (later referred to as content objects), and serves requests received from the front-end layer. Clients locate front-end layer nodes via DNS queries. As a result, ISPs can direct clients to the closest front-end layer nodes. Each DNS query is expected to return multiple front-end layer nodes. This enables clients to seamlessly switch from a failed to a working front-end layer node. Similarly, front-end layer nodes locate back-end layer nodes using their DNS name, thus achieving the same robust characteristics in the presence of back-end layer node failure.

Front-end Layer. The front-end layer nodes implement interfaces for application-layer protocols. For instance, a front-end layer node can implement an HTTP interface by listening on port 80, and converting GET-s and POST-s request to the corresponding back-end layer requests. Although in principle, different front-end layer nodes can implement different protocols, without loss of generality in the rest of the paper we assume that all nodes implement the same protocol.

Front-end layer nodes are analogous to NDN routers. To perform their functions, all front-end layer nodes implement the following two data structures: *the Forwarding Information Base* (FIB), and *the Pending Requests Table*, or PRT (similar to NDN's PIT). The FIB stores entries for each piece of content as several tuples $\langle namespace, (node_1, cost_1), \ldots, (node_n, cost_n)\rangle$, where $node_i$ indicates the IP address or DNS name of a node in the back-end layer which stores content with name starting with $namespace$, and $cost_i$ is a cost metric for requesting data to that node (e.g., round-trip time).

Upon receiving a request, a front-end layer host performs an FIB lookup using the name of the content being sent or received. The lookup returns a list of back-end layer nodes, each of which is suitable for addressing the request. If a front-end layer node forwards a request to a back-end layer node, and does not receive any response, the front-end layer node considers that the back-end layer node is unavailable, and updates the FIB accordingly. Periodically, the front-end layer node sends probe messages to a subset of the back-end layer nodes that store the content in to refresh $cost_i$ in the FIB. This mechanism ensures that DISPERSE is able to deliver content with low delay even when a large number of back-end layer nodes has failed. The PRT keeps track of requests that have been forwarded to back-end layer node layers, and for which the corresponding content has not been returned yet. Once a back-end layer node returns a content object, the front-end layer node performs a PRT look up to determine how the data should be converted to the appropriate protocol, and which host(s) requested the data. Upon correct match (or when they time out), entries are removed from the PRT.

Back-end Layer. The back-end is composed of nodes identified by a DNS name or IP address. A node can be a single host, or a collection of hosts (e.g., a data center). Nodes advertise their capabilities (e.g., bandwidth, available storage space), as well as their topology information and geographical location, to front-end layer nodes. When a back-end layer node receives new content, it sends a *namespace advertisement message* (containing one or more namespaces, common to the data objects received) to front-end layer nodes, to update their FIB.

We assume that the security properties of services offered using DISPERSE will vary widely. It is not reasonable to assume that all the nodes will faithfully enforce centralized fine-grained user-based access control policies on classified data. To address this challenge, DISPERSE builds on the access control and content authenticity mechanisms used in NDN. Specifically, it implements per-packet security: data objects are individually encrypted and authenticated (i.e., signed), and access control is implemented using cryptographic tools on the objects' decryption keys. This allows secure, efficient, and decentralized data protection. Access to the data therefore requires access to the decryption key, which is distributed using public key cryptography. For instance, decryption keys are encrypted under the public keys of the intended recipients, and stored as data.

Content in DISPERSE is replicated as content objects. Here, *content objects* can either mean different parts of content of a file or different files or different individual requests or different class of traffic requests as the network deems appropriate to implement. In general, large pieces of content (e.g., video files) are split into smaller content objects, which can be independently replicated and requested. This allows the replication algorithms to work at a fine granularity level, thus maximizing resource utilization and minimizing fragmentation. State explosion is prevented by grouping large sets of content objects into common namespaces. Due to the

absence of a central authority in charge of managing replication, unavailability of a subset of nodes does not prevent DISPERSE from making new copies of existing content. Strategies for replicating content are discussed in the following section.

IV. Randomized Node Failures

We first consider the case in which failures of back-end nodes are independent and identically distributed, i.e., nodes fail independent of each other, and probability of node failure is identical for all the nodes. We refer to this model as *randomized node failure model*.

Let $\mathcal{U}$ be the rate of upload (bandwidth) of content, and let $\mathcal{D}$ be the rate of download. (The notations used in the analysis in this section and their corresponding descriptions are provided in Table I.) In this paper, we assume that $\mathcal{D} \gg \mathcal{U}$, as this is a good representation of the behavior of popular applications such as the World Wide Web and YouTube [27].

We indicate the number of back-end layer nodes in the network with m, and the number of copies of each content object with c. Because each back-end layer node stores only one copy of each content object (storing multiple copies in the same node provides no additional resilience to node failures), we have $c \leq m$.

It is common for service providers to allocate more bandwidth to their servers than what is expected under average traffic conditions to account for *flash crowds* [28]. This also follows from the theory of large deviations of effective bandwidths, where the capacity of a network is larger than the expected traffic [29]. We therefore indicate the total bandwidth with $\lambda\mathcal{D}$, where $\lambda > 1$ is the over-provisioning factor.

Let $\mathcal{A}$ be amount of bandwidth that has become unavailable due to node failure. The remaining bandwidth in DISPERSE for retrieving content is $\lambda\mathcal{D} - \mathcal{A}$, and the portion of bandwidth available to retrieve a content object is:

$$P_{\text{available}} = \begin{cases} \lambda - \frac{\mathcal{A}}{\mathcal{D}} & \text{if } \mathcal{A} > \mathcal{D}(\lambda - 1) \\ 1 & \text{otherwise.} \end{cases} \quad (1)$$

The number of failed back-end layer nodes, m_f, can then be obtained as $m_f = \left\lceil \frac{\mathcal{A}}{\lambda\mathcal{D}} \cdot m \right\rceil$. Content can be retrieved from DISPERSE if *any one* of the $m - m_f$ nodes that has not failed holds the required content object. Thus, when all nodes fail with the same probability, independent of other nodes, the probability that content can be retrieved from any back-end layer node, P_{content} can be written as:

$$P_{\text{content}}(c) = \begin{cases} 1 - \frac{\binom{m_f}{c}}{\binom{m}{c}} & \text{if } c \leq m_f \\ 1 & \text{otherwise.} \end{cases} \quad (2)$$

Therefore, the probability of retrieving content from DISPERSE is:

$$P_{\text{disperse}} = P_{\text{available}} \cdot P_{\text{content}}(c). \quad (3)$$

Although increasing the number of content copies increases availability (see (2) and (3)), additional replication also leads to increased storage and bandwidth costs. In the rest of this section, we formally characterize the selection of the optimal number of copies (indicated henceforth as c^*) as an optimization problem. We first determine the amount of "utility" the network derives from each additional copy.

Let $D_{\text{net}} = \min(\mathcal{D}, \lambda\mathcal{D} - \mathcal{A})$. The utility, i.e., the net capacity available to the network to satisfy content requests from consumers is $\tilde{u}(c) = D_{\text{net}} \cdot P_{\text{content}}(c)$.

A. Without Delay Constraints

Because P_{content} is a monotonically increasing function of c, we have that $\tilde{u}(c)$ is also a monotonically increasing function of c. Therefore, without further constraints, DISPERSE network perceives maximum utility when infinite copies of the content are made. However, the network incurs a cost for making and storing each copy. This cost includes the storage, bandwidth, computation, and any other consideration involved in making new copies. We indicate this cost with α, and assume a linear cost model. (Other convex models for cost are also possible, and we leave their treatment as future work.) As a result, the total cost incurred for making c copies of each content is αc. The network incurs additional fixed cost, c_{fixed}, that captures other network issues like infrastructure, setting up the front-end and back-end nodes, the links between the nodes in the network, etc. Therefore, the net utility function, $\overline{u}_{\text{net}}(c)$, can be written as:

$$\overline{u}_{\text{net}}(c) = \tilde{u}(c) - \alpha c = D_{\text{net}} \cdot P_{\text{content}}(c) - \alpha c - c_{\text{fixed}}. \quad (4)$$

Since the fixed cost, c_{fixed} does not depend on the decision variable, c, it can be omitted from the optimization formulation. The resulting *net utility* perceived by the network, $\tilde{u}_{\text{net}}(c)$, is then given by:

$$\tilde{u}_{\text{net}}(c) = \tilde{u}(c) - \alpha c = D_{\text{net}} \cdot P_{\text{content}}(c) - \alpha c. \quad (5)$$

The network must then determine the value of c^* that maximizes $\tilde{u}_{\text{net}}(c)$ in (5), i.e., which is optimal for the network. In (2) and (5), the optimization is with respect to c, which is an integer. Thus, maximization of $\tilde{u}_{\text{net}}(c)$ is an integer program, which is NP hard [30].

To make this problem tractable, we assume that $m \gg c$ as motivated, for example, in [4]. Examples of systems that satisfy this assumption include replica maintenance systems [31], which had about 500 nodes and 3 replicas of content and the Copysets system developed by Cidon *et al.* [32] which makes 3 replicas of content in a network of 5000 nodes. A more recent article [33] also justifies this assumption. Further, we consider $m_f \gg c$, i.e., the number of failed back-end layer nodes is significantly larger than the number of copies of content in the system. If $m_f \ll c$, it corresponds to the case when the probability of node failure is very small. In this case, all copies available at any back-end node are accessible irrespective of the number of copies of content made by the network. Further, if $c < m_f$ (with $c \approx m_f$), then the network can always make additional $m_f - c$ copies to ensure that content is available. Content replication implementations in [31] and [32] showed cases when the number of failed nodes is about 60 % of the total number of nodes (which was about 300 in [31] and 3000 in [32]), while the number of replicas were 3. Essentially, the network must determine the optimal c^* only when $m_f \gg c$ so that the solution to making optimal amount

TABLE I
NOTATION USED IN THIS PAPER.

Notation	Description	Notation	Decription
m	Number of back-end layer nodes	m_f	Number of failed back-end layer nodes
$\mathcal{U}$	Rate at which content is uploaded	$\mathcal{D}$	Rate at which content is retrieved
c	Number of copies	c^*	Optimal number of copies
λ	Over provisioning factor	$\mathcal{A}$	Failed bandwidth in DISPERSE
P_{content}	Probability of content retrieval from DISPERSE back-end nodes	P_{disperse}	Probability of content retrieval in DISPERSE
$P_{\text{available}}$	Fraction of the bandwidth available to retrieve a content object	α	Cost of storing unit content in any DISPERSE back-end layer node
$Z(c)$	Content retrieval delay from the DISPERSE back-end layer nodes to the DISPERSE front-end layer nodes	$\tilde{u}(c)$	Benefit of storing c copies of content in DISPERSE back-end layer nodes
D_{net}	Available network bandwidth, defined as $D_{\text{net}} = \min(\mathcal{D}, \lambda\mathcal{D} - \mathcal{A})$	ρ	Cost of storing unit content in DISPERSE per unit of available bandwidth, defined as $\rho \triangleq \frac{\alpha}{D_{\text{net}}}$

of replicas of content is scalable. The integer constraint on c is first relaxed using the following theorem.

Theorem 1: Using Stirling's approximation (i.e., for any large integer K, we have $K! \approx \left(\frac{K}{e}\right)^K$ [34]), $\tilde{u}_{\text{net}}(c)$ can be written as:

$$\tilde{u}_{\text{net}}(c) = D_{\text{net}}\left(1 - p_f^{c-1}\right) - \alpha c. \tag{6}$$

Proof: Expanding (2) in terms of the factorials of the arguments,

$$P_{\text{content}} = 1 - \frac{m_f!}{(m_f - c + 1)!}\frac{(m - c + 1)!}{m!}. \tag{7}$$

Rewriting (7) using Stirling's approximation when $m \gg c$ and $m_f \gg c$,

$$P_{\text{content}} = 1 - \frac{m_f^{c-1}\left(1 - \frac{c-1}{m_f}\right)^{m_f - c + 1}}{m^{c-1}\left(1 - \frac{c-1}{m}\right)^{m-c+1}}. \tag{8}$$

Because $\frac{m_f}{m} = p_f$, and because for $m \gg c$ and $m_f \gg c$ we can apply $\lim_{k\to\infty}\left(1 - \frac{x}{k}\right)^k = e^{-x}$ [34] we obtain:

$$P_{\text{content}} = 1 - p_f^{c-1}. \tag{9}$$

This is used in (5) to obtain the objective function in (6). ■

Intuitively, (6) implies that in the randomized node failure scenario, when $m \gg c$ and $m_f \gg c$, the number of failed nodes is binomially distributed and hence, users fail to retrieve content when *all* the back-end layer nodes that have a copy of the content fail. The following theorem shows the uniqueness of c^*.

Theorem 2: There is a unique value of c^* that maximizes $\tilde{u}_{\text{net}}(c)$ in (6).

Proof: From (6),

$$\frac{\partial \tilde{u}_{\text{net}}}{\partial c} = -D_{\text{net}} \cdot p_f^{c-1} \ln p_f - \alpha, \tag{10}$$

and:

$$\frac{\partial^2 \tilde{u}_{\text{net}}}{\partial c^2} = -D_{\text{net}} \cdot p_f^{c-1} \left(\ln p_f\right)^2. \tag{11}$$

From the above, $\frac{\partial^2 \tilde{u}_{\text{net}}}{\partial c^2} < 0$ indicating that $\tilde{u}_{\text{net}}(c)$ is a concave function. Therefore, every local minimum obtained by solving the first order necessary condition:

$$\left.\frac{\partial \tilde{u}_{\text{net}}}{\partial c}\right|_{c=c^*} = -D_{\text{net}} \cdot p_f^{c^*-1} \ln p_f - \alpha = 0,$$
$$\text{i.e., } p_f^{c^*-1} = -\frac{\alpha}{D_{\text{net}} \cdot \ln p_f}, \tag{12}$$

is also a global minimum [35]. Moreover, the function, $\tilde{u}_{\text{net}}(c)$ is a continuously differentiable function [34] of c. Therefore, the solution obtained from the first order necessary condition in (12), is unique [35]. ■

From (12), the unique optimal value of c^* is obtained as:

$$c^* = \left\lceil 1 + \frac{\ln\left(-\frac{\alpha}{D_{\text{net}} \cdot \ln p_f}\right)}{\ln p_f} \right\rceil^+, \tag{13}$$

where $[y]^+ = \max(0, y)$ for any real number y. To make an integer optimal number of copies, we take the ceiling of the optimal solution in (13). The following observations can be made from (13) about c^*:

Observation 1) $\frac{\partial c^*}{\partial \alpha} = \frac{1}{\alpha \ln p_f} < 0$, since $p_f < 1$ (and hence, $\ln p_f < 0$), when D_{net} and p_f are fixed, i.e., c^* decreases as α increases, i.e., DISPERSE must maintain fewer copies. This is intuitively true because the network makes fewer copies as the cost of making a copy increases.

Observation 2) $\frac{\partial c^*}{\partial D_{\text{net}}} = -\frac{1}{D_{\text{net}} \cdot \ln p_f} > 0$. In other words, c^* increases as D_{net} increases. Intuitively, this indicates that more copies of the content can be made when more net bandwidth (i.e., more resource) is available.

Next, we determine the range for α such that DISPERSE performs a meaningful number of copies. In particular, we are interested in determining the upper limit $\alpha_{\max}$ on the cost of making a copy, beyond which DISPERSE has no incentive in making any copy of the content. The following theorem addresses this problem.

Theorem 3: Let $\alpha_{\max} \triangleq -D_{\text{net}} \cdot p_f \ln p_f$. Then, $\forall\ \alpha > \alpha_{\max}$, no copies of content are made.

Proof: Since $p_f < 1$ and hence, $\ln p_f < 0$, $\alpha_{\max} > 0$. When $\alpha = \alpha_{\max}$, $c^* = 0$, from (13). Since c^* decreases when α increases (from **Observation** 1) mentioned above), we have that $\forall\ \alpha > \alpha_{\max}$, $c^* = 0$. ■

Conversely, we would like to identify a lower limit $\alpha_{\min}$ to the cost of replicating content that the network must incur, such that $\alpha \leq \alpha_{\min}$ makes the optimal number of copies degenerate. The next theorem addresses this problem.

Theorem 4: $\exists\ \alpha_{\min} > 0$ such that the optimization solution in (13) is valid only for $\alpha > \alpha_{min}$.

Proof: The optimization of c^* requires that $m \gg c$ and $m_f \gg c$. Therefore, $\exists\ c_{\max}$ so that m_f is comparable to c, $\forall\ c > c_{\max}$. Let the value of α that yields $c = c_{\max}$ from (13) be $\alpha_{\min}$. As shown earlier, as α decreases, c^* increases, i.e., $c^* > c_{\max}$ if $\alpha < \alpha_{\min}$, thus rendering the optimization solution degenerate. ■

B. Adding Delay Constraints

In this subsection, we establish how additional constraints on the content retrieval delay incurred by DISPERSE affect the optimal number of content copies. To this end, we consider the delay from a front-end layer node to the i^{th} back-end layer node to be X_i, $1 \leq i \leq c$ (we only consider back-end layer nodes that store the required content). The optimization problem associated with the value of c^* is therefore subject to the constraints that the average delay, henceforth indicated as $\overline{Z}(c)$, is below a specified threshold τ. $Z(c)$ is given by $Z(c) = \min_{i=1}^{c} X_i$, since the front-end layer node retrieves content from the first back-end layer node with content that responds, and $\overline{Z}(c)$ by $\overline{Z}(c) = E[Z(c)]$.

Since delay represents an additional constraint for the problem of finding the optimal number of copies, the network might have to make additional copies in order to satisfy all constraints. Theorem 5 confirms this intuition. Before we present Theorem 5, we introduce the following two lemmas.

Lemma 1: Consider a sequence of real numbers $y_1, y_2, \ldots, y_k$. Let $\tilde{y}^{(k)} = \min_{i=1}^{k} y_i$. Then $\tilde{y}$ is a non-increasing function of k.

Proof: Let $\tilde{y}^{(k)} = \min_{i=1}^{k} y_i$. Then $\tilde{y}^{(k+1)} = \min_{i=1}^{k+1} y_i = \min(\tilde{y}^{(k)}, y_{k+1})$. Either $\tilde{y}^{(k+1)} = \tilde{y}^{(k)}$ (if $\tilde{y}^{(k)} \leq y_{k+1}$) or $\tilde{y}^{(k+1)} = y_{k+1}$ (if $\tilde{y}^{(k)} > y_{k+1}$). Therefore, in both cases $\tilde{y}^{(k+1)} \leq \tilde{y}^{(k)}$. Hence, $\tilde{y}$ is a non-increasing function of k. ■

Lemma 2: $\overline{Z}(c)$ is a non-increasing function of c.

Proof: If c is an integer, the proof follows from Lemma 1. For real values of c, $Z(c) = Z(\lceil c \rceil)$ Therefore, if $c_1 > c_2$, $\lceil c_1 \rceil \geq \lceil c_2 \rceil$ and hence, $Z(c_1) \leq Z(c_2)$, from Lemma 1 and hence, $\overline{Z}(c_1) = E\left[Z(c_1)\right] \leq \overline{Z}(c_2) = E\left[Z(c_2)\right]$. ■

Let $\hat{c}$ be the optimal number of copies made by the network under delay constraints. Then $\hat{c}$ is the solution to the following optimization problem:

$$\hat{c} = \arg\max_{c} \tilde{u}_{\text{net}}(c), \tag{14}$$

where $\tilde{u}_{\text{net}}(c)$ is define by (5), and subject to the constraint $\overline{Z}(c) \leq \tau$. The following theorem not only verifies the intuition that the network has to make additional copies of the content to satisfy the delay constraints, but also provides the quantitative value of the number of copies.

Theorem 5: The value $\hat{c}$ that maximizes $\tilde{u}_{\text{net}}(c)$, defined by (5) subject to the delay constraint $\overline{Z}(c) \leq \tau$, is:

$$\hat{c} = \max(c^*, \tilde{c}), \tag{15}$$

where $\tilde{c}$ is the value of c that satisfies $\overline{Z}(\tilde{c}) = \tau$.

Proof: Let $\tilde{c} \leq c^*$. Then by Lemma 2, $\overline{Z}(c^*) \leq \overline{Z}(\tilde{c}) = \tau$, thus satisfying constraint $\overline{Z}(c) \leq \tau$. Since c^* maximizes the objective function $\tilde{u}_{\text{net}}(c)$ from (12),

$$\hat{c} = c^*, \text{ when } \tilde{c} \leq c^*. \tag{16}$$

Let $\tilde{c} > c^*$. Then, from Lemma 2, $\overline{Z}(c^*) > \tau$, violating constraint $\overline{Z}(c) \leq \tau$. Then $\hat{c} = \arg\max_{c \geq \tilde{c}} \tilde{u}_{\text{net}}(c)$.

From (10), $\frac{\partial \tilde{u}_{\text{net}}(c)}{\partial c} > 0$ (i.e., $\tilde{u}_{\text{net}}(c)$ is an increasing function of c) for $c < c^*$ and $\frac{\partial \tilde{u}_{\text{net}}(c)}{\partial c} < 0$ (i.e., $\tilde{u}_{\text{net}}(c)$ is a decreasing function of c) for $c > c^*$. Therefore, the value of c that maximizes $\tilde{u}_{\text{net}}(c)$ for $c \geq \tilde{c} > c^*$ is

$$\hat{c} = \tilde{c}, \text{ when } c^* < \tilde{c}. \tag{17}$$

Theorem 5 follows by combining (16) and (17). ■

Theorem 5 implies that the number of copies must increase as the delay constraint becomes more stringent (i.e., as τ decreases). However, from (6), $c \gg c^*$ implies that $\tilde{u}_{\text{net}}(c)$ decreases. Therefore we next determine whether a stringent delay constraint can make the optimal number large to the point that the results is negative net utility for the network. This is an important issue, because under these conditions the network should not make copies of the content. The following theorem addresses this issue.

Theorem 6: $\exists\ \tau_{\min} > 0$ such that if $\tau < \tau_{\min}$, then the optimal strategy for the network is not to make any new copies.

Proof: Let $\tilde{u}_{\text{net}}(c^*) < 0$. The highest net utility experienced by the network is negative, i.e., the cost incurred by the network in making additional copies outweighs the utility perceived by the network. As a result, the network has no incentive in making additional copies, even when there is no delay constraint, i.e., when $\tau \to \infty$. Therefore, here $\tau_{\min}$ is a positive real number.

Let $\tilde{u}_{\text{net}}(c^*) > 0$. From (10), $\frac{\partial \tilde{u}_{\text{net}}(c)}{\partial c} < 0$ (i.e., $\tilde{u}_{\text{net}}(c)$ is a decreasing function of c) for $c > c^*$. Let c_{zero} be the value of c so that $\tilde{u}_{\text{net}}(c_{\text{zero}}) = 0$. Let $\tau_{\min} = \overline{Z}(c_{\text{zero}})$. Therefore, for $\tau = \tau_{\min}$, $c^* = c_{\text{zero}}$ and hence, when $\tau < \tau_{\min}$, $c^* > c_{\text{zero}}$, by Theorem 5. Then $\tilde{u}_{\text{net}}(c^*) < 0$. On the other hand, $c = 0$ implies $\tilde{u}_{\text{net}} = 0$. Therefore, the network is better off not making any copies of the content. ■

C. Numerical Evaluation of the Optimal Number of Copies

In this subsection, we compute the optimal number of content copies in DISPERSE, based on the analysis presented above. Fig. 2 shows the optimal number of copies made as a function of $\rho \triangleq \frac{\alpha}{D_{\text{net}}}$, i.e., ρ is the ratio of the cost per unit of available bandwidth. We consider three different orders of magnitude of ρ: (1) $\rho \sim 10^{-6}$, in Fig. 2(a); (2) $\rho \sim 10^{-4}$, in Fig. 2(b); and (3) $\rho \sim 10^{-3}$, in Fig. 2(c). Figs. 2(a)-2(c) are generated using $m = 3000$ back-end layer nodes. The optimal number of copies is between 5 and 15, while the number of failed nodes is $m_f = 300$, 600, 900, 1200, when $p_f = 0.1$, 0.2, 0.3, 0.4, respectively, justifying the assumption, $c \ll m_f$, used in (6). Figs. 2(a)−2(c) also indicate that the optimal number of copies is mainly affected by the probability of node failure, and just marginally by ρ. For instance, for $\rho = 5 \cdot 10^{-6}$, c^* is 15, 12, 10, and 7 for $p_f = 0.1$, 0.2, 0.3, and 0.4, respectively. When $\rho = 5 \cdot 10^{-4}$, c^* is 10, 9, 7 and 5, for $p_f = 0.1$, 0.2, 0.3, and 0.4, respectively. Finally for $\rho = 5 \cdot 10^{-3}$, c^* is 7, 6, 4, and 3, for $p_f = 0.1$, 0.2, 0.3, and 0.4, respectively. Essentially

for three orders of magnitude of decrease in ρ, the number of copies increases by a $2\times$ factor. Thus, the optimal number of copies is far more sensitive to p_f than to ρ, justifying the need for an architecture resilient to node failures as proposed in this work.

D. Comparison with CDN

To compare the performance of DISPERSE with that of CDN, we measure two parameters: (i) the percentage of requests for which the network was unable to deliver the content and (ii) the delay in retrieving content evaluated as the average number of nodes the network must query to retrieve the content. The percentage of failed content retrieval requests in DISPERSE is evaluated as $(1 - P_{\text{disperse}}) \cdot 100$, where P_{disperse} is obtained from (3). To determine the percentage of failed requests in CDNs, we proceed as follows. We evaluate the optimal number of copies, c^* for DISPERSE (from (13)). Since each copy of the content is hosted in a different server, we consider c^* number of CDN servers and assume the probability of node failure, p_f, to be identical both for CDN as well as for DISPERSE (so that we ensure we compare networks with identical properties).

There are currently several large deployments of CDNs, which differ in several architectural and design choices (see, e.g., [36]). In order to present the effectiveness of the proposed architecture, we choose to compare with Akamai [4]-[6], which is one of the most popular CDNs [37]. In Akamai, first the network attempts to retrieve content from a CDN server and if that attempt fails, then it attempts to retrieve it from the origin server. If that attempt also fails, then it is considered as a failure in retrieving content. The network does not attempt to retrieve content from other CDN servers by the very nature of the design of CDNs and instead directly chooses to retrieve content from the origin server [3], [38]-[40].

Thus, the network fails to receive content if it fails to retrieve content from a CDN server (given by the $\binom{c^*}{1}\frac{1}{c^*}p_f$ term in (18)) *and* it fails to retrieve it from the origin server (given by the p_{origin} term in (18)). Typically, the origin server is backed up against failures better than CDN servers by techniques like IP Anycast [41], [42]. Therefore, we take p_{origin} to be on order of magnitude less than the probability of failure of a CDN server, p_f. The probability of failure to retrieve content in CDN, P_{CDN}, is then given by:

$$P_{\text{CDN}} = \binom{c^*}{1} \cdot \frac{1}{c^*} p_f \cdot p_{\text{origin}} = p_f p_{\text{origin}}. \tag{18}$$

Fig. 3(a) shows the comparison of the percentage of content that the network fails to recover in CDN (taking $p_{\text{origin}} = \frac{p_f}{10}$) and in DISPERSE, which shows that the DISPERSE architecture results in a reduction by about 5 orders of magnitude, providing improved resilience to node failures.

We then proceed to study the tradeoff that DISPERSE has to bear to provide the advantage of improved resilience. For DISPERSE, the average number of back-end nodes that must be queried to retrieve content is:

$$\begin{aligned}\overline{N}_{\text{disperse}} &= \sum_{k=1}^{c^*} k(1-p_f)p_f^{k-1} + c^* p_f^{c^*} \\ &= \frac{1-(c^*+1)p_f^{c^*} + c * p_f^{c^*+1}}{1-p_f}.\end{aligned} \tag{19}$$

For CDNs, the number of queried nodes is 1 if the CDN server hosting content does not fail and 2 if the CDN server hosting content fails (irrespective of what happens to the origin server). Therefore, the average number of nodes queried in CDN,

$$\overline{N}_{\text{cdn}} = 1 - p_f + 2p_f(1-p_f) + 2p_f^2 = 1 + p_f. \tag{20}$$

Fig. 3(b) depicts the average delay (in terms of average number of nodes queried by the network to retrieve content) for different node failure scenarios. As observed from the figure, the delay is lower for CDN than that for DISPERSE. This is because, in CDNs query a maximum of two nodes (the CDN server and the origin server). However, DISPERSE queries as many nodes until it reaches a back-end node that hosts the content and has not failed.

However, Fig. 3(b) indicates that for low values of node failures (up to $50-60$ % node failires), DISPERSE queries only one additional node to retrieve content, compared to CDNs. For larger values of node failures (around 90%) the delay in DISPERSE increases by one order of magnitude compared to CDN. However, from Fig. 3(a), at this network condition, CDN fails to recover content with a significantly high probability while DISPERSE recovers content almost surely. Therefore DISPERSE provides high resilience to node failures without significantly increasing the delay in retrieving content.

The discussions in this subsection are based on a randomized node failure model where all nodes have same failure rate and same cost of replicating different content and all content have equal priority. The next section discusses optimal content replication in a setting where nodes have different failure rates and contents have different priorities.

V. Targeted Node Failures

When different content objects have different priorities and value, and when nodes have different amount of resources allocated to them, they are also likely to fail with different probabilities. For example, an adversary might focus on attacking specific nodes, rather than random subsets of nodes. We call this setting, *targeted node failure* scenario, where in, the benefit of storing content j in node i is given by:

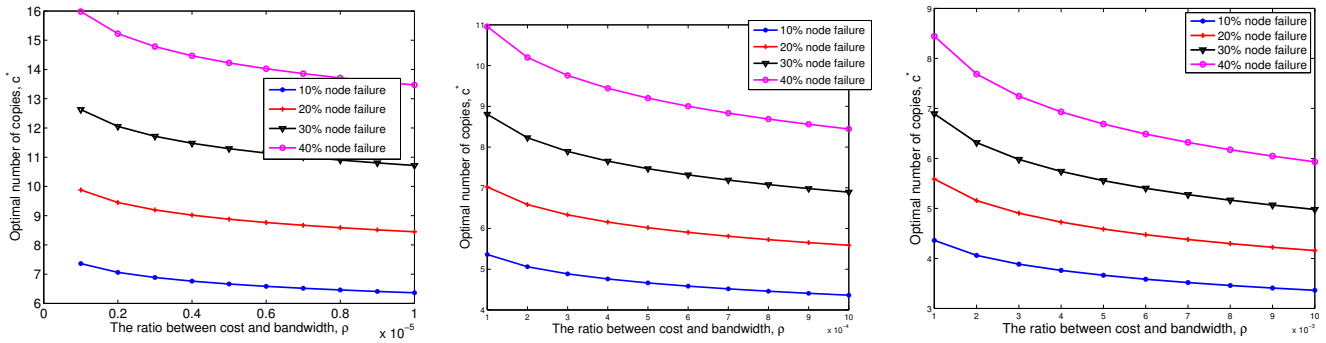
$$\tilde{u}_{ij}^{\text{net}} = D_{\text{net}}^{(j)} \cdot \left(1 - p_f^{(i)}\right) - \alpha_{ij}, \quad \begin{matrix} 1 \le j \le k, \\ 1 \le i \le m, \end{matrix} \tag{21}$$

where $D_{\text{net}}^{(j)}$ represents the benefit of storing content object j which, in turn, includes the "importance", or value, of content object j. In (21), $p_f^{(i)}$ is the probability that node i fails, and α_{ij} is the cost of storing content object j in node i. Let content j utilize x_{ij} amount of resources in node i, X_i be the total amount of resources available at node i and $\mathcal{S}_j$ be the set of nodes that hold content j. Then, the objective is:

$$\text{Maximize over all } \mathcal{S}_j, \sum_{j=1}^{k} \sum_{i \in \mathcal{S}_j} \tilde{u}_{ij}, \tag{22}$$

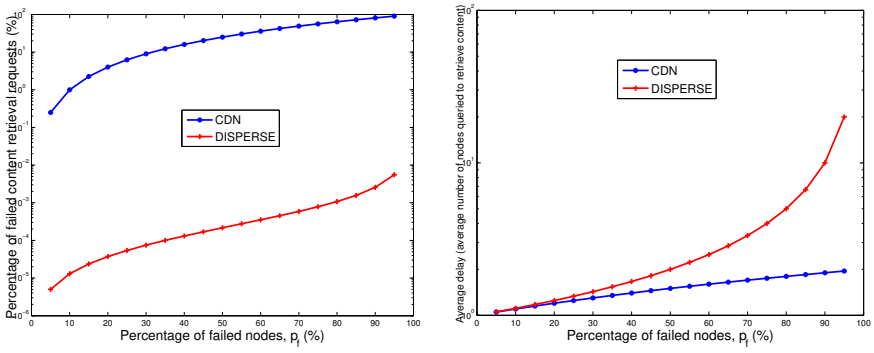
$$\text{subject to } \sum_{j=1}^{k} x_{ij} \le X_i, \ 1 \le i \le m, \tag{23}$$

which is an NP-hard problem in general [30]. However, in our setting this problem can be reduced to a minimum-cost network



(a) $\rho \sim 10^{-6}$ (b) $\rho \sim 10^{-4}$ (c) $\rho \sim 10^{-3}$

Fig. 2. Optimal number of copies made in the DISPERSE back-end layer nodes, for the randomized node failure model, for $m = 3000$ nodes. The ratio between cost and bandwidth, ρ, is varied to different orders of magnitude. Results show that the optimal number of copies is sensitive to node failure probability, p_f, and less sensitive to ρ, as change in two orders of magnitude in ρ change the optimal number of copies by 2–5. The optimal number of copies $c^* \sim$ 5–15, while the number of failed nodes is $m_f = 300$, 600, 900, and 1200, for $p_f = 0.1$, 0.2, 0.3, and 0.4, respectively, justifying the assumption that $c \ll m_f$.



(a) Content Retrieval (b) Delay

Fig. 3. Comparison of the performance of DISPERSE and CDN in terms of failure to retrieve content and delay in retrieving content. While computing delay, we evaluate the number of nodes queried by the network to retrieve the content. This also includes requests timeout.

flow problem [43]. The minimum cost flow approach has been used before as a solution for secure multipath routing [44], and to find a minimum cost set of nodes to execute a false data injection attack [45]. However, to our knowledge, we are the first to leverage efficient solutions for the minimum cost flow approach to content replication.

We instantiate the minimum-cost network flow problem on the bi-partite graph [46] depicted in Fig. 4. One set of nodes represent content (red nodes in Fig. 4), while another set of nodes represents DISPERSE back-end layer nodes (blue nodes in Fig. 4). Nodes s and t are dummy source and sink nodes. Edges from s to content nodes have infinite capacity and zero cost. Edges from content nodes to DISPERSE back-end layer nodes have cost $\alpha_{ij} - D_{\text{net}}^{(j)} \cdot \left(1 - p_f^{(i)}\right)$, and infinite capacity. Edges from back-end layer nodes to the sink have cost zero, and capacity X_i. The edge from the sink to the source has cost $-\infty$, to avoid the degenerate solution of all flows being zero (see [43] for details). Content j is considered to be stored in node i if the flow in edge (j, i) is positive.

Theorem 7: Content object j is stored in node i if and only if there is a flow augmenting path through edge (j, i).

Proof: If content object, j is assigned node i, then a positive flow is added on edge, (j, i), indicating that the path $s \to j \to i \to t$ is a flow augmenting path.

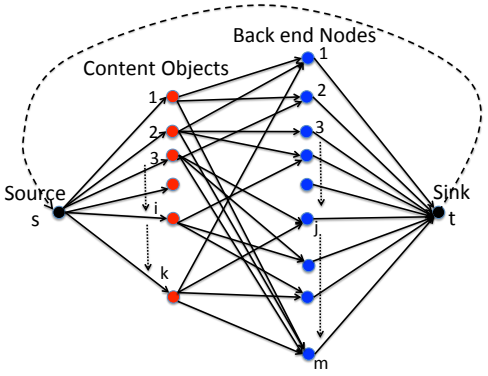
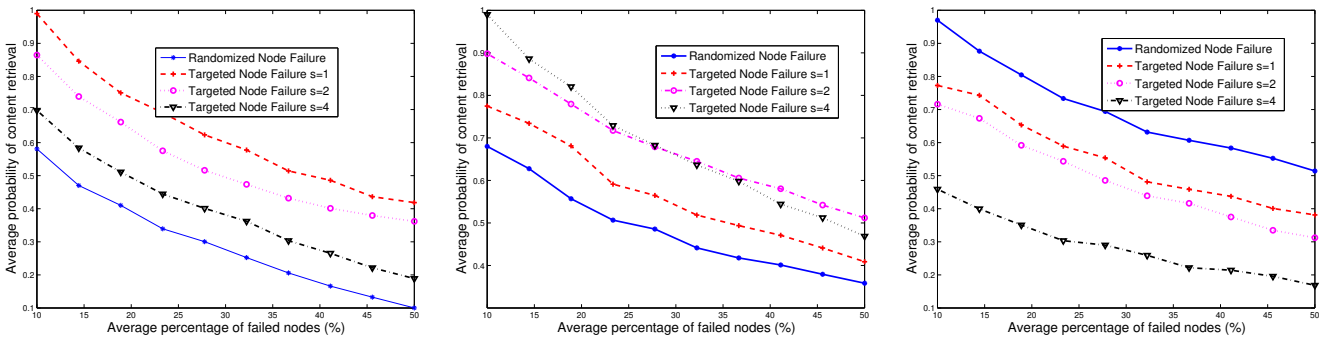


Fig. 4. Network model used to solve objective function (22), subject to constraints in (23). Node s is a dummy source, and node t is a dummy sink.

If there is a flow augmenting path from $s \to j \to i \to t$, then a positive flow is added on the edge, (j, i) indicating that content object, j is stored in node, i. ■

When some nodes fail, or if new content objects arrive, then flows are re-assigned on edges to incorporate the new content object or to account for changing node failures and costs. A re-assignment is equivalent to adding negative flow. For instance, in Fig. 4, let content object 3 be assigned to node 2. If node



(a) All Content (b) Highest Priority Content (c) Lowest Priority Content

Fig. 5. Performance of DISPERSE for targeted node failures when content objects of all priorities are equally likely. The legends, $s = 1$, $s = 2$ and $s = 4$ represent the values of the $s-$parameter for the Zipf distribution of the node failures.

2 fails then the content in content object 3 is re-assigned to another node by identifying a flow augmenting path from s to t passing through content object 3. This is equivalent to first assigning a negative flow on the edge from content object 3 to node 2 and adding the same amount of positive flow from content object 3 to the new node it is assigned to. The cost for the negative flow is also negated to ensure the same value of the cost-flow product. The next two theorems identify the maximum number of reassignments.

Theorem 8: Let f be a current cost flow in the network and let f^* be the optimal cost flow. Then, $f^* - f$ can be decomposed into a set of at most $|E|$ negative cost cycles, where $|E|$ is the number of edges in the network.

Proof: Consider any feasible flow f, and the optimal flow f^*. The optimal flow vector is obtained from the feasible flow vector by saturating unsaturated edges. Also, to satisfy flow conservation constraints, it is essential to find a negative cost directed cycle and modify the flow by the same amount on all edges on the cycle. Hence f^* can be obtained from f by a sequence of identifying negative cost directed cycles and saturating at least one edge in each cycle. The flow $f^* - f$ can be seen as a feasible flow in the residual graphs, which can also be decomposed into a sequence of negative cost directed cycles saturating at least one edge at a time. Since there are $|E|$ edges, the flow $f^* - f$ can be decomposed into at most $|E|$ negative cost directed cycles. ■

Theorem 9: There are at most $\lfloor \frac{|E|}{2} \rfloor$ re-assignments.

Proof: Let the number of negative cost directed cycles be Ω. It is noted that for the problem under consideration, a negative cost directed cycle can be of length 4 to 6. For a cycle of length, γ, $\gamma/2$ edges in the cycle correspond to a re-assignment. Therefore, C number of cycles can result in $\lfloor \frac{C}{2} \rfloor$ re-assignments, which, from Theorem 8, is $\lfloor \frac{|E|}{2} \rfloor$. ■
From the algorithms specified in Chapter 9 in [43], the complexity of the minimum cost flow problem for a system with k conent objects and m nodes is $O\left((k+m)^3 \log(k+m)\right)$.

A. Numerical Evaluation of the Optimal Number of Copies

To evaluate the performance of the minimum cost flow approach, we perform simulations on Ubuntu Linux platform. We generate a network of $m = 3000$ nodes and $k = 100$ content objects. Let r_j be the priority index of content object, j. This is an integer in the set, $\{1, 2, \cdots, 10\}$ where in, a larger index represents a content object with higher priority. Content objects are assigned priority indices, r_j, as follows. We use the built-in random number generator in the library, which is uniformly distributed in $[0, 1]$. This then is multiplied by 10 and the final floating point number is truncated to obtain an integer between 0 and 9. Then 1 is added to make the integer take values between 1 and 10.

The failure probabilities of nodes, $p_f^{(i)}$, $1 \leq i \leq m$ are generated according to a Zipf distribution [47], i.e., $p_f^{(i)} = 1 - \frac{\frac{1}{i^s}}{\zeta(s)}$, where $\zeta(s) = \sum_{j=1}^{m} \frac{1}{j^s}$ is the Riemann's Zeta function [34]. Our simulations are performed for $s = 1$, $s = 2$ and $s = 4$. The average node failure probability is defined as the average of all $p_f^{(i)}$'s.

The cost, α_{ij}, $1 \leq i \leq m$, $1 \leq j \leq k$ is generated by generating random numbers uniformly distributed in $\left[0, 10^{-6}\left(1 - p_f^{(i)}\right) r_j\right]$. This not only ensures that content objects with higher priority incur higher average costs than those with low priority when stored in the same node, but also ensures that the same content object incurs higher cost when stored in a node which is less likely to fail than when stored in one that is more likely to fail.

We then run 100,000 simulation experiments by repeating instances of the random numbers generated as described above. For each instance, we randomly permute the order of the failure probabilities of nodes using the random permutation algorithm in [**?**] so that successive simulations create significantly different scenarios. Content objects are then stored in different nodes according to our minimum cost flow optimization framework. Then, node failures are generated as follows: For each node, i, a random number is generated using the built-in library (which, by default, is uniformly distributed in $[0, 1]$). A node, i, is considered to "fail" if the generated random number is below $p_f^{(i)}$. Once a node fails, re-assignments are performed according to the cost flow framework. Failure of content retrieval is said to occur if a content object in a failed node cannot be re-assigned to another node.

Fig. 5 shows the performance of the minimum cost flow algorithm for the targeted node failure scenarios in comparison with the randomized node failure scenarios. We compared the content retrieval probabilities for all content types (Fig. 5(a)),

content with the highest priority (Fig. 5(b)) and content with the lowest priority (Fig. 5(c)). As shown in Fig. 5(a), at 30% node failure scenario, the minimum cost flow approach results in a content retrieval probability of 0.3 for the homogenous network and 0.39, 0.54 and 0.69 for $s = 1$, 2, and 4, respectively, leading to an improvement of $1.3\times$ to $2.3\times$. When s increases, the probability of retrieving content decreases. This is because, as s increases, some nodes are less likely to fail, and therefore all content objects compete to be stored in the same node, However, limited resource availability forces content to be stored in nodes that are more likely to fail, thus decreasing the the probability of content retrieval. All these scenarios perform better than the randomized node failure case, because we store content in a failure-aware manner.

We then proceed to study the impact of the minimum cost flow approach on the highest and lowest priority content objects. For highest priority content object (Fig. 5(b)), as s increases the probability of content retrieval increases. This is because high priority content is stored in nodes that are less likely to fail, and therefore more likely to be retrieved. However, the difference between benefit and cost is not monotonic with respect to s. Fig. 5(b), shows that up to a node failure probability of 25–30%, $s = 4$ leads to better performance than $s = 2$. However, when node failure is above 30%, $s = 2$ leads to better performance. This is because while the network attempts to store highest priority content in nodes with the lowest node failure probabilities, resource constraints associated with high failure rates result in high priority content stored in nodes with higher failure probabilities. The increase in the content retrieval probability results in a trade-off or penalty in the form of increase in loss of content for low priority content types, as observed from Fig. 5(c). As expected, larger values of s result in worse performance of content retrieval for low priority content type, because this content is stored in nodes with higher probabilities of failure.

We finally proceed to study two other cases of distribution priorities for content objects, in addition to the uniform distribution studied thus far: (1) There are more low priority content objects than high priority content objects. To achieve this, we generate content object with priority, $r_j \in \{1, 2, \cdots, 10\}$, with probability, $\frac{11-r_j}{1+2+3+\cdots+10} = \frac{11-r_j}{55}$. Since we consider lower index, r_j to indicate lower priority, this generates high priority objects with lower probability. (2) Then we generate higher priority content objects with higher probability. To achieve this, we generate content objects with priority $r_j \in \{1, 2, \cdots, 10\}$ with probability, $\frac{r_j}{55}$. Since higher value of r_j represents a higher priority, this method generates high priority content objects with higher probability.

Fig. 6 shows the performance of DISPERSE with targeted node failures when the amount of low priority content is larger (Fig. 6(a)) and when amount of high priority content is larger (6(b)). As observed from Fig. 6(a), when low priority content is more likely, DISPERSE with a node failure probability according to a Zipf distribution with $s = 1$ still out-performs the content replication strategy adopted for the network with randomized node failures. Specifically, when 40% nodes fail, the content replication strategy discussed in Section IV (i.e., assuming randomized node failures) yields a content retrieval probability of 0.52, whereas, for the targeted node failure scenario with $s = 1$, the content retrieval probability is about 0.78, which is an improvement of 50% (or a factor of $1.5\times$). However, when more nodes fail for $s = 2$ and $s = 4$. When content with different priority are placed in different nodes (according to their failure probabilities and costs), high priority content can still be recovered successfully but low priority content is lost. Since there is more low priority content over all, the content retrieval probability decreases and is lesser compared to assigning content without consideration to the different node failure probabilities and content priorities.

The nature of the flow maximization algorithm discussed in this section, enhances availability of high priority content. Therefore, when there is more high priority content, it results on a larger content retrieval probability, over all (as observed from Fig. 6(b)). For an average node failure rate of 40%, the content replication mechanism using the non-linear optimization problem discussed in Section IV (i.e., assuming randomized node failures) results in a content retrieval probability of about 33%, where as, this increases to about 83% when deploying the flow maximization algorithm discussed in this section. This is an increase by a factor of $2.5\times$.

VI. Conclusion

We introduced DISPERSE, a novel distributed architecture for content replication that provides resilience against node failures. DISPERSE reduces the failure of content retrieval by five orders of magnitude compared to CDNs, without significantly increasing content retrieval delay. Our analysis shows that the optimal number of content replicas is more sensitive to node failures than to resource availability and cost of storage. We consider this a strong justification for DISPERSE. When content was hosted in nodes according to their priorities, resources availability, and susceptibility of the node to failure, content availability further increases by a factor of $1.3\times$ to $2.3\times$. Specifically, the content availability for high priority content was increased, with a trade-off of reduced availability of low priority content.

Our future research directions include addressing issues of cache replacement and cache consistency. These problems have been addressed in various contexts, including in NDN (see, e.g., [48], [49]), on which DISPERSE is based. Although these techniques are directly applicable to DISPERSE, further investigation can be done to optimize these approaches for node failure scenarios.

References

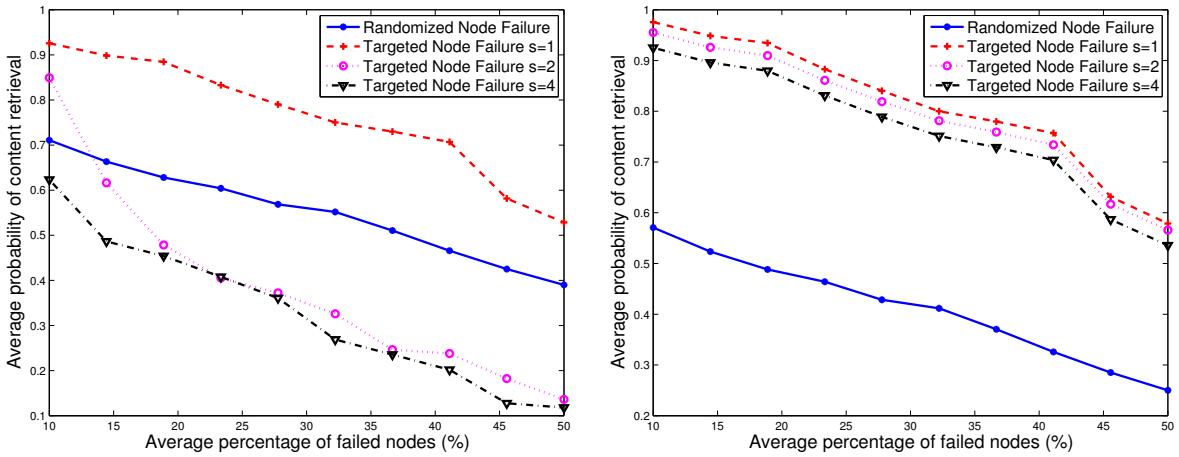
[1] A. K. Pathan and R. Buyya, "A taxonomy and survey of content delivery networks," *Grid Computing and Distributed Systems Laboratory, University of Melbourne, Technical Report*, p. 4, 2007.

[2] G. Pallis and A. Vakali, "Insight and perspectives for content delivery networks," *Commun. of the ACM*, vol. 49, no. 1, pp. 101–106, 2006.

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

[4] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai network: A platform for high-performance Internet applications," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, pp. 2–19, 2010.

[5] (2017, May) Akamai for responsive web design. [Online]. Available: https://www.akamai.com/us/en/multimedia/documents/infographic/akamai-for-responsive-web-design-infographic.pdf



(a) More low priority content

(b) More higher priority content

Fig. 6. Content retrieval probability in DISPERSE with targeted node failures. Content objects with different priorities occur according to a non-uniform distribution. The legends, $s = 1$, $s = 2$ and $s = 4$ represent the values of the $s-$parameter for the Zipf distribution of the node failures.

[6] Akamai, "CDN architecture." [Online]. Available: https://www.akamai.com/us/en/resources/cdn-architecture.jsp

[7] Y. Kitamura, Y. Lee, R. Sakiyama, and K. Okamura, "Experience with restoration of asia pacific network failures from taiwan earthquake," *IEICE Trans. on Commun.*, vol. 90, no. 11, pp. 3095–3103, 2007.

[8] (2014, Jan) Ea login servers experience ddos attack, origin offline (update) 101. [Online]. Available: https://www.polygon.com/2014/1/2/5268652/origin-servers-experience-ddos-attack

[9] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, D. Massey, C. Papadopoulos *et al.*, "Named data networking (NDN) project," *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, 2010.

[10] J. Kangasharju, J. Roberts, and K. W. Ross, "Object replication strategies in content distribution networks," *ACM Jl. on Computer Commun.*, vol. 25, no. 4, pp. 376–383, Mar. 2002.

[11] S. U. Khan and I. Ahmed, "Internet content replication: A solution through game theory," *Technical Report, Computer Science and Engineering, CSE-2004-5*, Jul. 2004.

[12] G. G. Pollatos, O. A. Telelis, and V. Zissimopoulos, "On the social cost of distributed selfish content replication," *Intl. Conf. on Research in Networking (ICRN'2008)*, May 2008.

[13] S. Ganguly, A. Saxena, S. Bhatnagar, S. Banerjee, and R. Izmailov, "Fast replication in content distribution overlays," *Proc., IEEE Intl. Conf. on Computer Commun. (INFOCOM'2005)*, Mar. 2005.

[14] C. La, P. Michiardi, C. Casetti, and C. C. M. Fiore, "Content replication in mobile networks," *IEEE Jl. on Sel. Areas in Commun.*, vol. 30, no. 9, pp. 1762–1770, Oct. 2012.

[15] W. Wu and J. S. C. Li, "Exploring the optimal replication strategy in P2P-VoD systems: Characterization and evaluation," *IEEE Intl. Conf. on Computer Commun. (INFOCOM'2011)*, Mar. 2011.

[16] B. J. Ko, V. Pappas, R. Raghavendra, Y. Song, R. B. Dilmaghani, K. Lee, and D. Verma, "An information-centric architecture for data center networks," *Info. Centric Networking (ICN'2012)*, Aug. 2012.

[17] S. Moharir and N. Karamchandani, "Content replication in large distributed caches," Mar. 2016. [Online]. Available: https://arxiv.org/abs/1603.09153

[18] Z. Li and G. Simon, "In a Telco-CDN, pushing content makes sense," *IEEE Trans. on Network and Service Mgmt.*, vol. 10, no. 3, pp. 300–311, Sep. 2013.

[19] A. Aral and T. Ovatman, "A decentralized replica placement algorithm for edge computing," *IEEE Trans. on Network and Service Mgmt.*, vol. 15, no. 2, pp. 516–529, Jun. 2018.

[20] B. Oh, S. Vural, and N. W. R. Tafazolli, "Priority-based flow control for dynamic and reliable flow management in SDN," *IEEE Trans. on Network and Service Mgmt.*, vol. 15, no. 4, pp. 1720–1732, Dec. 2018.

[21] W. K. Lin, C. Ye, and D. M. Chiu, "Decentralized replication algorithms for improving file availability in P2P networks," *Intl. Workshop on Quality-of-Service (IWQoS'2007)*, Jun. 2007.

[22] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, and B. Li, "Design and deployment of a hybrid cdn-p2p system for live video streaming: experiences with livesky," in *Proceedings of the 17th ACM international conference on Multimedia*. ACM, 2009, pp. 25–34.

[23] C. Huang, A. Wang, J. Li, and K. W. Ross, "Understanding hybrid cdn-p2p: why limelight needs its own red swoosh," in *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, 2008, pp. 75–80.

[24] S. Seyyedi and B. Akbari, "Hybrid cdn-p2p architectures for live video streaming: Comparative study of connected and unconnected meshes," in *Computer Networks and Distributed Systems (CNDS), 2011 International Symposium on*. IEEE, 2011, pp. 175–180.

[25] Z. Lu, Y. Wang, Y. R. Yang *et al.*, "An analysis and comparison of cdn-p2p-hybrid content delivery system and model." *JCM*, vol. 7, no. 3, pp. 232–245, 2012.

[26] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Commun. Mag.*, vol. 50, no. 7, 2012.

[27] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: a view from the edge," in *Proc., ACM SIGCOMM'2007*, 2007.

[28] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal, "Dynamic provisioning of multi-tier internet applications," in *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*. IEEE, 2005, pp. 217–228.

[29] A. Kumar, J. Kuri, and D. Manjunath, *Communication Networks: An Analytical Approach*. Morgan Kauffman, 2004.

[30] T. Cormen, C. Leisserson, R. Rivest, and C. Stein, *Introduction to Algorithms*. McGraw Hill, 2009.

[31] B. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. F. Kaashoek, J. Kubiatowicz, and R. T. Morris, "Efficient replica maintenance for distributed storage systems," *Networked Systems Design and Implementation (NSDI'2006)*, May 2006.

[32] A. Cidon, S. Rumble, R. Stutsman, S. Katti, J. Ousterhout, and M. Rosenblum, "Copysets: Reducing the frequency of data loss in cloud storage," *USENIX'2013*, Aug. 2013.

[33] M. Kleppmann, "The probability of data loss in large clusters," Jan. 2017. [Online]. Available: https://martin.kleppmann.com/2017/01/26/data-loss-in-large-clusters.html

[34] W. Rudin, *Principles of Mathematical Analysis*. McGraw Hill, 1976.

[35] H. Varian, *Intermediate Microeconomics: A Modern Approach*. W. W. Norton and Co., 1996.

[36] B. M. Moreno, C. E. P. Salvador, M. E. Domingo, I. A. Pena, and V. R. Extremera, "On content delivery network implementation," *Elsevier Computer Commun.*, vol. 29, no. 12, pp. 2396–2412, Dec. 2006.

[37] A. Salvarinov, H. Borenstein, V. O. Odeyemi, M. Deluca, K. Yu, and A. Asghar, "Implementation of a content delivery network (CDN)," *IBM Caching Strategy Design for WebSphere Commerce Series, Part 1*, Sep. 2014. [Online]. Available: https://www.ibm.com/developerworks/library/co-wc-caching-part1/index.html

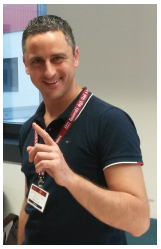
[38] J. Dunn and B. Crosby, "What your CDN won't tell you: Optimizing a news website for speed and stability," *Proc., USENIX Symposium on Internet Technologies and Systems (USENIX'2012)*, Jun. 2012.
[39] X. Jiang and J. Bi, "nCDN: CDN enhanced with NDN," *Computer Commun. Workshops (INFOCOM WORKSHOPS'2014)*, Mar. 2014.
[40] B. M. Maggs and R. K. Sitaraman, "Algorithmic nuggets in content delivery," *ACM Computer Commun. Review*, vol. 45, no. 3, pp. 52–66, Jul. 2015.
[41] [Online]. Available: https://learn.akamai.com/en-us/webhelp/alerts/alerts/GUID-FCD6293F-6EBC-45A4-9172-4EE4EC2CA2DE.html
[42] [Online]. Available: https://www.cloudflare.com/learning/cdn/cdn-load-balance-reliability/
[43] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms and Applications*. Prentice Hall Inc., 1993.
[44] P. P. Lee, V. Misra, and D. Rubenstein, "Distributed algorithms for secure multipath routing," in *Intl. Conf. on Computer Commun. (INFOCOM'2005)*, Mar. 2005.
[45] O. Kosut, "Max-flow min-cut for power system security index computation," in *IEEE Sensor Array and Multichannel Signal Processing Workshop (SAM'2014)*, 2014.
[46] F. Harary, *Graph Theory*. Addison-Wesley, 1969.
[47] S. M. Ross, *A First Course in Probability*. Pearson, 2013.
[48] D. E. Knuth, *The Art of Computer Programming: Volume 1*. Addison-Wesley, 1968.
[49] M. Xie, I. Widjaja, and H. Wang, "Enhancing cache robustness for content-centric networking," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 2426–2434.
[50] M. Conti, P. Gasti, and M. Teoli, "A lightweight mechanism for detection of cache pollution attacks in named data networking," *Computer Networks*, vol. 57, no. 16, pp. 3178–3191, 2013.

Santhanakrishnan Anand received his Ph.D. degree from the Indian Institute of Science, Bangalore, India, in 2003. He received the best Ph.D. dissertation medal in the electrical sciences division. He is currently an Assistant Professor at Department of Electrical and Computer Engineering, New York Institute of Technology. His current research includes spectrum management and security in next generation wireless networks, covert timing channels and information propagation in Internet media. He represented Samsung in 3GPP SA2, IEEE 802.16j and 802.20 forums.

Ding Ding received the Ph.D. degrees in computer science at University of Padua, Italy, under the supervision of Prof. Mauro Conti, in 2017. After that, he was a research fellow in New York Institute of Technology, U.S., worked with Prof. Paolo Gasti. His research interests include network security and distributed networks.

Paolo Gasti is an Associate Professor of Computer Science at the New York Institute of Technology (NYIT), School of Engineering and Computing Sciences. His work focuses on behavioral biometrics, privacy-preserving biometric authentication, secure multi-party protocols, and network security. His research has been sponsored by the National Science Foundation and the Defense Advanced Research Project Agency. He received his B.S., M.S., and Ph.D. degrees from University of Genoa, Italy. Before joining NYIT, he worked as a research scholar at University of California, Irvine. He is a former Fulbright scholar, and member of the IEEE. He directs NYIT's Laboratory for behavioral Authentication, Machine learning, and Privacy (LAMP).

Mike O'Neal is a professor and former Chair of the Computer Science Program at Louisiana Tech University, where he holds the Larson Endowed Professorship. Mike has three decades of experience in the field of higher education, has co-founded two high tech startups over the past two decades, and has fifteen issued US patents to his name. Dr. O?Neal received his BS (Magna Cum Laude, 1982) and MS (1984) from Louisiana Tech University, and his Ph.D. (1989) from the University of Louisiana, Lafayette. In the 1999 to 2001 time frame Mike was Co-Founder and CTO of OneNetNow.com, a community-based web portal focused on the urban community, which was acquired by EarthLink in 2001. From 2001 to 2012 Mike was Co-Founder and CTO of Network Foundation Technologies, a company focused on distributed broadcast technologies. Dr. O?Neal?s academic research interests include active authentication using behavioral biometrics, computer science education, artificial intelligence, and distributed online broadcast technologies. Dr. O?Neal is currently working with a Fortune 200 company to develop and commercialize keystroke based continuous authentication – a system that verifies your identity by the way you type.



Mauro Conti is an Associate Professor at the University of Padua, Italy. He obtained his Ph.D. from Sapienza University of Rome, Italy, in 2009. After his Ph.D., he was a Post-Doc Researcher at Vrije Universiteit Amsterdam, The Netherlands. In 2011 he joined as Assistant Professor the University of Padua, where he became Associate Professor in 2015. In 2017, he obtained the national habilitation as Full Professor for Computer Science and Computer Engineering. He has been Visiting Researcher at GMU (2008, 2016), UCLA (2010), UCI (2012, 2013, 2014, 2017), TU Darmstadt (2013), UF (2015), and FIU (2015, 2016).

He has been awarded with a Marie Curie Fellowship (2012) by the European Commission, and with a Fellowship by the German DAAD (2013). His research is also funded by companies, including Cisco and Intel. His main research interest is in the area of security and privacy. In this area, he published more than 200 papers in topmost international peer-reviewed journals and conference. He is Associate Editor for several journals, including IEEE Communications Surveys & Tutorials, IEEE Transactions on Information Forensics and Security, and IEEE Transactions on Network and Service Management. He was Program Chair for TRUST 2015, ICISS 2016, WiSec 2017, and General Chair for SecureComm 2012 and ACM SACMAT 2013. He is Senior Member of the IEEE.

Kiran S. Balagani is currently an Associate Professor of Computer Science with the New York Institute of Technology. He directs NYIT's Laboratory for behavioral Authentication, Machine learning, and Privacy. His research interests are in cyber-behavioral anomaly detection (e.g., unauthorized user-access behaviors), behavioral biometrics, and privacy-preserving biometrics. His research has been sponsored by the National Science Foundation and Defense Advanced Research Project Agency. His teaching interests include development of graduate and undergraduate courses in network security, biometrics, and machine learning. He received the Ph.D. degree from Louisiana Tech University, USA.