UNIVERSITY OF CALIFORNIA

SANTA CRUZ

SOCIAL NETWORK CODING RATE CONTROL IN INFORMATION CENTRIC DELAY TOLERANT NETWORKS

A thesis submitted in partial satisfaction
of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

SAMUEL BENNETT WOOD

December 2014

The Thesis of Samuel Wood
is approved:

Professor J.J. Garcia-Luna-Aceves, Chair

Professor Katia Obraczka

Professor Brad Smith

Tyrus Miller
Vice Provost and Dean of Graduate Studies

UMI Number: 1583279

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

UMI 1583279

Published by ProQuest LLC (2015). Copyright in the Dissertation held by the Author.

Microform Edition © ProQuest LLC.
All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

Copyright © by

Samuel B. Wood

2014

Table of Contents

List of Figures

List of Tables

Abstract

Social Network Coding Rate Control
in Information Centric Delay Tolerant Networks
by
Samuel Bennett Wood

Tactical and emergency-response networks require efficient communication without a managed infrastructure in order to meet the requirements of mission critical applications. In these networks, mobility, disruption, limited network resources, and limited host resources are the norm instead of the exception. Despite these constraints, applications must quickly and reliably share data collected from their environment to allow users to coordinate and make critical decisions. Our previous work demonstrates that applying information-centric paradigms to the tactical edge can provide performance benefits over traditional address centric approaches. We expand on this work and investigate how social relationships can be inferred and exploited to improve network performance in volatile networks.

As a result of our investigation, we propose SOCRATIC (SOCial RATe control for Information Centric networks), a novel approach to dissemination that unifies replication and network coding, which takes advantage of social content and context heuristics to improve network performance. SOCRATIC replicates network encoded blocks according to a *popularity index* metric that captures social relationships, and is shared during neighbor discovery. The number of encoded blocks that is relayed to a node depends on its interest in the data object and its popularity index, i.e., how often and for how long it meets other nodes. We observe that nodes with similar interests tend to be co-located and we exploit this information through use of a generalization of a data object-to-interest matching function that quantifies this similarity. Encoded blocks are subsequently replicated towards the subscriber if a stable path exists. We evaluate an implementation of SOCRATIC through a detailed network emulation of a tactical scenario and demonstrate that it can achieve better performance than the existing socially agnostic approaches.

Dedication

To my parents,

Sam Wood and Eve Bennett-Wood,

for their unceasing support and courage,

when times got curiouser than I *could* suppose.

Intelligence is the capacity to receive, decode and transmit information efficiently. Stupidity is blockage of this process at any point.
— Robert A. Wilson

Acknowledgments

This work could not have been completed without the help of my adviser, Professor J.J. Garcia-Luna-Aceves, whose ingenuity and positivity serves as a constant resource for inspiration. Professor Sadjadpour's guidance and inventiveness served as the catalyst for this research. I thank my colleagues at the CCRG Lab and at UCSC, especially James Mathewson and Michael Cutter for their perseverance and dedication to knowledge and truth. This work is partially based on the DARPA CBMEN project—in particular, it is based on Joshua Joy, Dr. Mark-Oliver Stehr, and Professor Gerla's work in network coding in the context of that project.

The text of this thesis includes reprint of the following previously published material:

> Samuel Wood, Hamid Sadjadpour, J.J. Garcia-Luna-Aceves, "SOCRATIC: A Social Approach to Network Coding Rate Control." To appear in IEEE GLOBECOM 2014.

The co-authors listed in this publication directed and supervised the research which forms the basis for the thesis.

1 Introduction

Disaster response and tactical networks require efficient delay-tolerant communication without any fixed infrastructure guarantees. With the advent of inexpensive mobile computers (e.g., personal data assistants, laptops, cell phones, and wearable technology) and inexpensive vehicle platforms (e.g., quad-copters, remote control helicopters, planes, and cars), new kinds of heterogeneous mobile networks are forming that use inexpensive commodity hardware in tactical and disaster response scenarios [3]. As an example, suppose a large earthquake occurs which damages communications infrastructure (e.g., down cell and ham radio towers), as well as underground infrastructure (e.g., cut coaxial and fiber cables) and power. A city might call upon a volunteer communications disaster response team to survey damage and aggregate data into reports that are delivered to a local emergency response commission responsible for resource allocation to reestablish critical communications infrastructure for firefighters and police. It is envisaged that squads of disaster response members will travel together using either their own personal mobile devices or mobile devices provided by the city to collect data such as video, photos, voice, and textual reports which are shared with other squad members and used for inter-squad coordination. Efficient and opportunistic communication with limited resources is paramount for the safety of each squad member and for the success of the mission. Temporary infrastructure such as throwboxes [48] and unmanned vehicles are deployed at will to increase network capacity, and mobile devices must automatically use these resources without user configuration. An important aspect to these networks is that almost every node works in concert to accomplish a common goal, where each node may have different roles and responsibilities in accomplishing the overarching goal. This assumption is relaxed in the tactical case, since adversaries may actively jam the network or commandeer devices with sensitive information.

The key research question that this thesis addresses is whether social information (e.g., the roles and responsibilities of the nodes) can be used by disaster response networks to improve network performance. Our conclusion is yes, and this thesis proposes and evaluates SOCRATIC, a socially aware method of content dissemination that improves performance over existing socially agnostic approaches in certain disaster networks. With this approach, new social constructs do not need to be introduced into the network through mechanisms such as explicit or learned social network graphs, which require either modifications at the application layer (increase in deployment effort) or a learning phase (increase in delivery delay). Indeed, SOCRATIC uses existing network primitives (publications and subscriptions) to infer meaningful relationships which directly influence content dissemination for improved performance.

The remainder of this section details the preliminary research, definitions, and motivation. Chapter 2 discusses the related work. Chapter 3 details the SOCRATIC algorithm and provides an idealized scenario that exercises all of its functionality. Chapter 4 describes the test emulation system in detail, and describes the evaluation scenario. Chapter 5 presents the performance results which illustrate the benefits SOCRATIC has over several existing dissemination approaches. Chapter 6 summarises the findings of this research and concludes with future work.

1.1 Preliminaries

Although mobile devices come with increasingly larger amounts of storage, processing power, and communication interfaces, efficient and reliable inter-device communication remains as a challenging research problem and it is sometimes refereed to as the "last tactical mile" problem [19, 21]. Traditional host-centric networking abstractions (e.g., TCP over IP) are not disruption tolerant, having been designed for host-to-host communication in a connected static network, with short round trip times between hosts, per-packet switching of fixed length messages, bidirectional connectivity with mostly symmetric rates, and low packet loss. The plethora of mobile ad-hoc networks (MANET) research has uncovered efficient routing protocols for large scale mobile networks that are mostly connected though dynamic topologies (relaxing the static assumption of traditional networks), however a great body of this research is not immediately applicable to networks where there is frequently never a contemporaneous path between the communicating parties, or if the path is only in one direction, or significantly delayed.

1.2 Delay-tolerant Networks

As an alternative to Internet and MANET research, delay-tolerant (or disruption tolerant) networks (DTN) is a broad research area that explores dissemination and replication strategies for communication where there may never be a contemporaneous path between hosts. As described in [13, 46], DTNs use a sender-driven communication primitive that names an individual destination host using a universal resource indicator. DTN takes a “store, carry, and forward” approach to routing where a variable length message (termed a bundle) is forwarded on a hop-by-hop basis and is temporarily cached (with a bundle time-to-live, TTL, optionally specified by the sender) at relay nodes until it is delivered to the destination. Bi-directional communication, contemporaneous paths, and short delays are not assumed. Security policies operate on the bundle as opposed to the connection. Our research investigates a particular type of DTN, and how network protocols can maximize network performance.

Figure 1 illustrates a high-level diagram of both the traditional Internet architecture (the network stack), and a DTN stack. This diagram serves as a guideline of the technology interface boundaries, but alternative demarcations exist (e.g., 7 layer OSI model), and some technologies overlap layers. Each layer generally only interacts with its adjacent layer, through a well defined interface. For example, a Web application sends a message to a TCP socket which adds a TCP header and then encapsulate the message into an IP datagram. The application receives messages from the TCP socket which decapsulates the IP datagram and the TCP header. The data link and physical layers are typically the same in both the Internet and in DTNs, and they include technologies such as the IEEE 802.2 sub-layer, MAC sub-layer, wired Ethernet (IEEE 802.3), Bluetooth, USB, Wi-Fi (IEEE 802.11a/b/g/n/ac). The Internet network layer encompasses protocols such as IPv4, IPv6, ICMP and ARP, where routing tables are maintained by application layer protocols (e.g., OSPF, RIP, and EIGRP) and ARP tables (IP address to MAC address mappings) are maintained by the Address Resolution Protocol at the data link layer. IP functions as the so-called narrow-waist of the Internet, providing the global naming abstraction and the packet abstraction for data transport between autonomous system (AS) boundaries. Transport layer protocols for the Internet include UDP and TCP, where TCP provides a reliable, connection oriented interface of byte streams and features congestion control mechanisms, and UDP serves as an unreliable datagram delivery service for potentially lossy best effort communication. In a DTN, the network and transport layers may use the same Internet technologies and networks (e.g., as an overlay), but the key difference between the two architectures is the additional bundle layer abstraction which serves as the new narrow waist between DTN autonomous systems, or regions. Each DTN region may use different transport (and lower layer) protocols, and the bundle layer is responsible for storing and routing bundles on a hop-by-hop basis between DTN nodes, for the eventual delivery to the destination. The focus of our research is at the bundle layer.

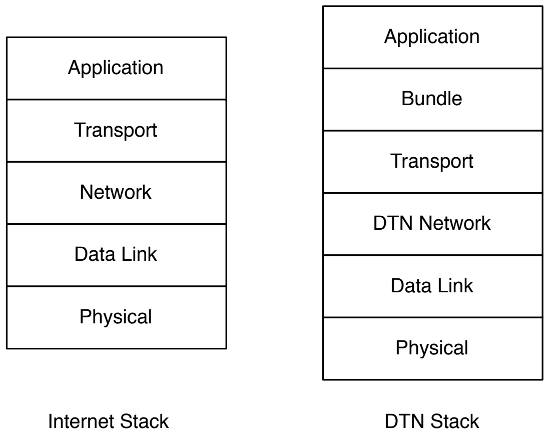

Figure 1: The network architecture diagrams for the traditional Internet and a DTN.

In delay-tolerant forwarding schemes the bundle replica is deleted upon successfully sending the bundle to a relay that is closer to the destination. As an alternative to forwarding techniques, *replication* is used to improve dissemination performance in networks with non-contemporaneous paths. With replication, nodes store replicas at relay nodes that need not be on the shortest path to the destination, with the intent that increasing the number of relays and replicas will increase the probability that the bundle will reach the destination. The research challenge for replication algorithms is how to judiciously select which relays receive the bundle, how many times to replicate the bundle, and how long a relay caches the bundle. The goal for dissemination is that the bundle is delivered with high probability with minimal consumption of network and device resources.

1.3 Information Centric Networks

Recent work demonstrates that the information centric networking (ICN) paradigm can improve content delivery and reduce latency in delay tolerant networks [29,30,33,37,47]. The ICN approach to networking changes the communication abstraction provided by the network layer from a socket between a pair of location-based addresses to a data object referenced by name [23,28,31], and is heavily influenced by the directed diffusion architecture for sensor networks [27]. ICN applications communicate with the network layer through an application programming interface (API) based on publish-subscribe primitives, where a publisher provides the content and its associated metadata to form a *data object*. We say that a subscription (or interest) *matches* a data object when it refers to the data object. There are several approaches to how metadata names a data object and each approach has its own advantages and disadvantages (see [22] for a discussion on this topic), but the approaches fall into three categories: 1) self-certifying flat names [31], 2) human readable hierarchical names [28], and 3) human readable attribute-value pairs [12,36]. The matching function is not necessarily bijective, and a single subscription may refer to multiple data objects and a single data object may match multiple subscriptions. There are two main approaches to the subscription life cycle: 1) persistent or long-standing (e.g., Haggle [36]), and 2) exact match (e.g., NDN [28]). In the persistent case, the subscription is affiliated with a TTL and may return multiple data objects that match during the specified time, while in the exact match case a single matching data object will "consume" the subscription and remove it from the matching node so that subsequent matching may not occur. Depending on the interest forwarding scheme, multiple data objects may be returned to a single exact match subscription.

The metadata is used by the subscriber to request the content and by the network to make caching and routing decisions for the data object. This approach enables communication over non-contemporaneous paths and supports ubiquitous caching: subscriber applications only specify *what* content they desire (not *where*), and the network decides *how* to deliver this content to the subscriber. Since a subscriber is unaware of the source of the content, any cast and nearest replica routing are supported. Data objects are cached opportunistically at intermediate relays along the path to a subscriber, according to a cache eviction policy. Publisher applications only specify *what* content they want to share, along with metadata to define a name that is used by the subscribers to reference the data object. Since the publisher does not know the subscribers at the time of publication, multicast communication is automatically supported without the need for the publisher to invoke a subscriber discovery protocol. Information Centric Delay Tolerant Networks (ICDTN) refers to the application of the ICN principles to DTNs [43]. Instead of the bundle layer providing only a send primitive for bundles, this layer is expanded to provide an ICN API. The main conceptual difference is that in ICDTN content discovery is the responsibility of the bundle layer. In other words, instead of the bundle layer routing bundles to hosts identified by URIs as in DTN, ICDTN routes subscriptions and data objects separately, in response to a data object match somewhere in the network (not necessarily at the publisher or subscriber). For example, in the Named Data Networking ICN architecture the Forwarding Information Base (FIB) is used to route interests, and the Pending Interest Table (PIT) is used to route content. The main challenge for ICDTNs is to efficiently deliver data objects to subscribers from caches and publishers. Given that global knowledge is unavailable, nodes must make local decisions based on limited network knowledge and intermittent neighbor connectivity.

1.4 Network Coding

Random linear network coding (RLNC) [24], or *network coding*, refers to a content dissemination technique that offers a generalization, and improvement, to fragmentation. At a high level, network coding uses cheap computational power to increase network efficiency. Fragmentation is a method to split a data object into smaller sized fragments (smaller data objects) that are each cached and routed independently, and the original data object is reconstructed and passed to the subscriber application upon the reception of all of the fragments. Fragmentation is employed when the connectivity duration between nodes is short, prohibiting the transfer of a large data object. In ICDTN, publishers do not necessarily know their subscribers during the life cycle of the data object, thus publishers may use a replication strategy for each fragment that is subscriber independent. Due to this lack of communication between the subscriber and the publisher, naive replication schemes for fragments may lead the subscriber to experience the classical "Coupon Collectors" problem from probability theory [18], forcing the subscriber to wait an inordinate amount of time for the last few missing fragments. Specifically, if the fragments are uniformly distributed for each cache that the subscriber meets, then the expected number of caches that the subscriber would need to visit to reconstruct the data object is $\Theta(m \log m)$ where m is the number of fragments. Among its benefits, random linear network coding lowers this bound to $\Theta(m)$ by adding a negligible amount of metadata for each fragment, and increasing the amount of processing per fragment at the subscriber. Additionally, it reduces the amount of communication necessary between hosts to determine the missing blocks: a fragmentation approach requires a mechanism (e.g., Bloom filters [10] or lists) to indicate what fragments a node already has and what fragments the node needs, while network coding does not require this communication—it only needs to indicate when it has received enough blocks.

Below we describe the RLNC encoding process as in [20]. When a publisher publishes a file M it subdivides it into a set of n equal sized fragments $M^1, \ldots, M^n$ (we set the fragment size to 32KB). The fragments are used to generate blocks $B^1, \ldots, B^k$, where each block B^j is the same size as a fragment, and is associated with with a sequence of coefficients $g^j = g_1^j, \ldots, g_n^j$ where each g_i^j is drawn uniformly at random from the finite field $\mathbb{F}_{p^s}$ (we set $p = 2$ and $s = 8$). B^j is generated by setting $B^j = \sum_{i=1}^{n} g_i^j M^i$ where the summation occurs over every symbol position in M^i. In other words, let B_m^j be the m-th symbol of B^j and M_m^i be the m-th symbol of M^i (encoded in $\mathbb{F}_{p^s}$), then $B_m^j = \sum_{i=1}^{n} g_i^j M_m^i$. We say that g^j is the *coefficient vector* for B^j, and B^j is the *information vector.* Instead of caching data objects containing fragments M^i, we cache data objects containing tuples (g^j, B^j). In a process called *mixing*, if a cache has received a set of tuples $\{(g^1, B^1), \ldots, (g^h, B^h)\}$, then it can generate new blocks H^i by picking a set of coefficients $o^i = o_1^i \ldots o_h^i$ uniformly at random from $\mathbb{F}_{p^s}$, and setting $H^i = \sum_{j=1}^{h} o_j^i B^j$, where the summation is over every symbol (as before). The coefficient vector $g^{i'} = g_1^{i'}, \ldots g_n^{i'}$ for H^i then becomes $g_k^{i'} = \sum_{j=1}^{h} o_j^i g_k^j$, forming tuple $(g^{i'}, H^i)$.

Once a subscriber has received a set of tuples $\{(g^1, B^1), \ldots, (g^k, B^k)\}$ where $k \geq n$, then it can attempt to use the blocks to decode the original data object. Specifically, the subscriber needs to solve the linear system of equations $\{B^j = \sum_{i=1}^{n} g_i^j M^i\}$ for $j = 1, \ldots, k$, with M^i unknown. Since the coefficients were selected randomly (possibly at different caches in the network), some of the blocks may be linearly dependent. The key property to enable decoding is that there are n linearly independent blocks. In practice, our selection of a large field $\mathbb{F}_{256}$ means that with high probability each newly generated block is linearly independent from the previously generated blocks, and Gaussian elimination is used to efficiently invert the matrix and decode. We use the term *rate* to refer to the number of encoded blocks k that the publisher generates. There is a trade off between the network coding rate, resource utilization, and delivery. In general, a higher rate improves delivery latency, but at a cost of increased network and processing usage, while a lower rate requires less network and processing utilization but reduces delivery latency. To see that network coding is a generalization of fragmentation, let $z^i = z_1^i, \ldots, z_n^i$ denote the coefficient vector where $z_j^i = 0$ for all $j \neq i$ and $z_i^i = 1$. Each coefficient vector in $z^1, \ldots z^n$ merely selects the i-th fragment when multiplied by the information vector, to form $B^i = M^i$.

1.5 Information CEntric Mobile Ad-hoc Networking (ICEMAN)

ICEMAN [29,33,47] was developed under the context of the DARPA Content-based Mobile Edge Networking (CBMEN) [3] project, and the result of this research was an evaluated ICDTN system which serves as a starting point and baseline for our investigation. We briefly describe the details of ICEMAN, as SOCRATIC is implemented as a module in this system. ICEMAN is a publish-subscribe system that is designed for situational awareness applications operating in networks subject to severe disruption. It adopts a declarative attribute-based approach to naming where applications request content or services by specifying attributes, their weights, a satisfaction threshold, and the number of content matches. This approach to naming unifies both exact match and long-standing queries, through a standard publish-subscribe API. It is an generalization of the Haggle architecture which itself is a solution for pocket switched networks (PSN) where communication occurs through point-to-point proximity encounters among two peers [25].

Figure 2 illustrates the ICEMAN architecture which SOCRATIC expands. Multiple applications connect to an ICEMAN daemon running on the local host through an API wrapper over TCP, so that libraries across a variety of languages are supported including C++, C, Java and Python, and TCP's congestion control mechanism is used to prevent the system from becoming overloaded servicing application requests. Within the user space daemon is a layer-less concurrent architecture where multiple managers maintain thread pools and communicate with one another by listening to and posting events on a shared event queue, in the so-called ICEMAN kernel. We have simplified the diagram for expository purposes by omitting some of the managers and submodules. We describe the managers as follows. The Application Manager is responsible for maintaining local application state, and interacts with the local applications by informing them when a data object match occurs and handling publications and subscriptions. Upon a data object match, the Forwarding Manager is triggered to determine what neighbor the data object should be replicated to in order to maximize the probability that the data object is received by the subscriber. The Cache Manager is responsible for maintaining the data object life cycle for previously received data objects, or data objects that have been published by a local application. It is also used to perform the matching function between data objects and interests. The Transport Manager is responsible for transmitting data objects between two hops and it is triggered by events created in the Forwarding Manager. Multiple transport protocols are supported, including TCP, UDP unicast, UDP broadcast, and Bluetooth RFCOMM. The Interface Manager is responsible for maintaining the list of local interfaces and 1-hop neighbor interfaces. This state is used by the Transport Manager to send the data over a network and supports technologies including Ethernet IEEE 802.3, WiFi IEEE 802.11a/b/g/n/ac, and Bluetooth. It informs the other managers when a new node is discovered or becomes disconnected by using a mechanism that is particular to each interface (e.g., periodic UDP broadcasts for WiFi with timeouts to determine disconnection).

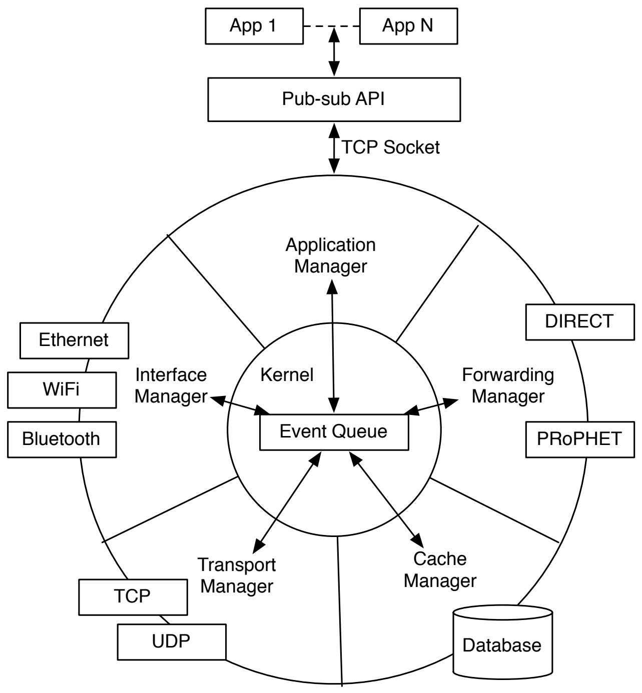

Figure 2: The layer-less ICEMAN architecture (which is itself based on the Haggle architecture) that SOCRATIC builds on. Managers run in separate threads and communicate to one another by posting and registering listeners to events on the daemon's main event queue.

In ICEMAN, a data object is the fundamental unit of abstraction, along with its metadata that is represented as a set of attribute-value pairs and a payload represented by a file. Every data object has a creation timestamp attribute that is assigned by the publisher. A *data object identifier*, or ID, is defined as the SHA-1 hash over all of this information, which is globally unique with high probability. A *node description* is a special type of data object that is used to propagate cache summaries and application interests. The subscriptions of multiple local applications are aggregated into a single interest node description and the local cache summary can optionally be included in this data object or propagated separately as a cache summary node description. Node descriptions are periodically disseminated over multiple hops and have a limited lifetime. Each node maintains the node descriptions of other nodes in the network, even if they are not neighbors. The cache summary uses a Bloom filter data structure to compactly represent an approximation of all of the data objects that are stored locally.[1]

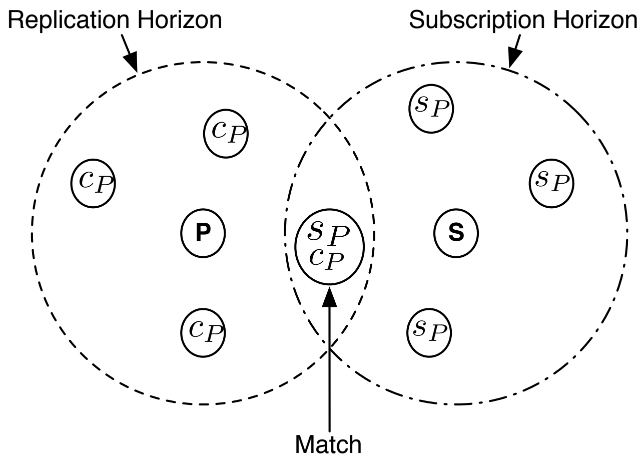

Figure 3: An illustration of the in-network subscription to publication matching. The solid circles represent nodes. Node "P" publishes a data object that is replicated to nearby nodes denoted c_P. Node "S" subscribes to this data object, and its interest is propagated to nodes denoted s_P. The dashed circles indicate the distance from the publisher (subscriber) that the data object (interest) has propagated.

Data object matching to interests occurs at a local cache whenever a new data object is received, whether it be a node description data object containing interests or

[1] A Bloom filter data structure [1] is a compact representation of a set that supports fast set membership addition and fast probabilistic membership queries. False negatives are not possible, but false positives are possible, usually with low probability—the fewer the elements the less likely for a false positive.

a file data object. Figure 3 illustrates how data objects and interests are propagated throughout the network, and shows how a match can occur at intermediate nodes who are neither publishers nor subscribers. If the data object is a node description and it is more recent (or fresh) than any previously received node description for its respective node, then each of the data objects in the local cache are checked to see if they match the newly received interests, and the node description is inserted into the cache (and any less recent node descriptions for that node are removed). If this data object contains a cache summary, then it is stored in a data base used by the Transport Manager to prevent unnecessary transmissions to neighbors, and it is used by the matching algorithm to avoid counting previously received data objects (as determined by the cache summary) as a match. If the node description is less recent than a previously received one for that node (this is checked using the timestamp attribute), then it is discarded. If the data object is not a node description, then each interest stored in the local cache is checked against the data object to see if there is a match.

When a data object is received, the Cache Manager determines whether or not the data object should be stored in the local cache by consulting a utility-based caching policy. Caching is employed to reduce latency and bandwidth use, and is required if there is no contemporaneous path between a source and a destination. ICEMAN approaches caching as an online utility maximization problem, similar to the cooperative caching method described by Chand et al [14] and the utility dissemination scheme in Spyropoulos et al [41]. A user specifies a policy through a function that is a composition of utility functions and can express both hard (evict immediately) and soft (keep if available resources) replacement policies. The caching manager identifies useful content and prioritizes evictions accordingly, least utility first. Each utility function assigns a real number between 0 (lowest utility) and 1 (highest utility) to each data object in the cache, which indicates the utility of storing that data object at that node. The functions are both content (e.g., a function of the data object's metadata or its payload size) and context sensitive (i.e., they vary in time and space). Data objects whose utility do not meet a minimum threshold are evicted immediately. Each function is multiplied by a constant weight factor between 0 and 1, where 0 disables the function. Once the cache exceeds a watermark capacity, the Cache Manager selects a set of data objects to evict to bring the cache capacity below the watermark. The minimum threshold, weight factors, and watermark capacity are all specified by the policy. The selection of which data objects to evict is framed as a 0-1 knapsack problem [15], where the watermark capacity is the bag size, the data object payload size is the cost, and the computed utility is the benefit. Since the optimization problem is NP-Hard, we use the greedy marginal utility heuristic to solve the problem in polynomial time. This approach computes a marginal utility as the benefit divided by the cost, and adds data objects greedily—greatest marginal utility—first to the bag until the bag size is exceeded. The remaining unselected data objects are evicted.

Although a complete description of the utility caching is outside the scope of this thesis, we briefly describe several utility functions to make the caching description concrete. The Relative Time-to-live (`RTTL`) function uses a specific attribute to determine how long a data object should be cached since its reception. If the node has cached the data object for less than this time then the utility is 1, otherwise it is 0. The Partial Order (`PO`) function uses specific attributes to define a partial order on classes of data objects. If the data object can be replaced by another data object in the local cache using the partial order, then the utility is 0, otherwise it is 1. The utility functions are composed to create complex policies using the `sum`, `min`, and `max` utility functions, which compute the sum, min and max of its constituent utility functions, respectively. Each utility function returns a real within $[0, 1]$ for each data object d based on the state of node i at time t, s_t^i. An example policy is

$$U_1(d, s_t^i) = min\{RTTL(d, t), PO(d, s_t^i)\}. \quad (1)$$

With this policy suppose that the threshold is set to some number greater than 0, then any data object that exceeds the relative time-to-live or is eligible for replacement is evicted. This specific policy is used internally in ICEMAN and SOCRATIC for node descriptions, where the relative time-to-live is 30 seconds and the partial order is over the value of the creation time attribute in milliseconds since the Unix epoch.

When a match occurs, an event is raised by the Forwarding Manager to determine to which neighbor the data object should be replicated so that it will be delivered to the subscriber with high probability. The dissemination policy can be a function of the data object's metadata, or the same mechanism is used for all data objects. DIRECT, PRoPHET (described in Chapter 2) and NONE are example dissemination mechanisms, where NONE only replicates to a neighbor if that neighbor has an application that is actively subscribed to the data. The Forwarding Manager may decide whether to generate network coded blocks or fragments depending on content and context specific policies. In this case, block (fragment) data objects are generated which inherit all of the attributes as the original, or *parent*, data object, along with additional attributes to enable decoding, such as the parent's ID and payload size. Instead of sending the parent data object, the Forwarding Manager sends all of the blocks (fragments) to the selected neighbor.

After a neighbor has been selected to receive a data object, the content transport policy is consulted by the Transport Manager to determine how the data object should be sent to the neighbor. Before the data object is passed to the transport layer to send to the neighbor, the cache summary of the receiving node is consulted to determine if the node already has it stored locally. If the data object is a block (fragment) then both the block and the parent ID are checked in the cache summary. An optional control protocol may sit above the underlying transport protocol to further reduce redundant transmissions. In TCP, the control protocol first sends the data object metadata to the receiver, who may send back an acknowledgement or a reject message. If an acknowledgement is received, then the sender sends the data object payload. Upon successfully sending a data object to a neighbor, the sender preemptively updates the neighbor's cache summary to reflect that it was received—this cache summary is overridden when a more recent one from the neighbor is received using the partial order utility function and creation timestamp. If the data object being sent was a block (fragment), then the reject message may indicate whether the receiver has the parent data object, in which case the cache summary is preemptively updated to reflect the parent ID. In addition to TCP, UDP unicast and UDP broadcast are supported. With UDP broadcast, the cache summary of all of the neighbors are preemptively updated.

1.6 Towards Social Rate Control

At a high level, our research focuses on efficient content dissemination in disaster response and tactical scenarios consisting of several small groups (e.g., 10 node squads) that are persistently connected within several hops. For brevity and clarity purposes, we will consistently use standard military terminology. Inter-squad communication is typically non-contemporaneous and can require message ferrying through members of another squad, or using a UAV. We observe that members within the same squad are co-located and are likely subscribed to similar content. By definition, squads are cohesive units that are located within the same region at the same period of time, and squad members are interested in content generated by other members in their squad as well as content that is pertinent to the specific region in which they are operating. As an example tactical scenario, when a squad moves to to a new geographic area an ICDTN application will automatically subscribe to spot and BOLO[2] reports that are generated by other squads who either currently exist in that area, or previously visited that geographic area. This subscription identifies reports for the region by name (e.g., a neighborhood name) or by using GPS coordinate ranges. Conversely, as nodes change missions they will un-subscribe to content that is no longer relevant to the new mission. Since missions are specific to geographic areas, we observe that nodes with similar subscriptions will tend to be co-located. Nodes may have similar but different hardware capabilities and roles. For example, UAVs cover a much wider geographic area and have more power and weight resources than a squad member. Squad leaders will tend to meet

[2]BOLO is an acronym used in the Department of Defense for Be On the Look Out, and refers to reports that are issued to identify and apprehend a wanted person. From the Oxford Dictionary of the US Military, a spot report is: "A concise narrative report of essential information covering events or conditions that may have an immediate and significant effect on current planning and operations."

other squad leaders more often than non-squad members of another squad, due to hierarchies established by military doctrine.

The research question becomes how can the social structure inherent to ICDTNs (due to nodes working in concert to accomplish a common goal) be quantified and exploited to improve content dissemination performance? In particular, can subscriptions be used, and if so how can subscription similarity information be quantified and used? SOCRATIC (SOCial RATe control for Information Centric networks) is one approach to this problem that uses replication heuristics based on social content and context information to improve network performance. Unlike previous approaches, SOCRATIC replicates a bounded number of network coded blocks according to the satisfaction degree of the data object and the potential relay, along with other social-context heuristics. SOCRATIC adopts ICEMAN's naming scheme where an application subscribes to data objects by specifying a human readable list of attributes and a matching threshold: this scheme generalizes exact match and computes a *satisfaction degree* for a $\langle$ node, data object $\rangle$ pair. It is implemented as a module within the Forwarding Manager and Caching Manager (see Section 1.5), and replicates blocks in response to discovering new neighbors or receiving new data objects through local publications. In contrast with traditional replication approaches that operate on a data object level, SOCRATIC intelligently replicates a bounded number of network-coded blocks that are subsequently replicated. SOCRATIC enforces a type of rate control that limits the total number of generated network-coded blocks and imposes a dynamic limit for each relay. A publisher for a data object decides to replicate a bounded number of coded blocks to each encountered neighbor based on the neighbor's *popularity index.* Nodes that frequently visit new neighbors, visit neighbors for long periods of time, or have more interest in the data object, have a higher popularity index and receive more blocks. Using this criteria, SOCRATIC can achieve better network performance than approaches to replication that are socially-agnostic.

2 Related Work

We briefly describe related DTN approaches to content dissemination. Epidemic dissemination [45] is one baseline for DTN routing protocols: in its simplest form, nodes exchange cache summaries and replicate each missing data object so that their caches are identical. To improve epidemic dissemination, PRoPHET [34] reduces the number of replicas by sharing encounter history summaries among nodes and by transitively building an encounter probability matrix; this results in replication of content to nodes that have a high probability of meeting the destination. Bounded replica approaches such as Spray and Wait [38] enforce an explicit constraint on the number of data object replicas to finely control network usage; relays may further replicate the data object so long as the bound is not exceeded (e.g., using *forwarding tokens*), and relay selection is based on first encounter. Utility-based DTN approaches [9, 41] select relays and data objects for replication according to a generic utility function, and may also support bounded replicas or a utility threshold. Typical utility functions aim to reduce delivery latency and network usage by selecting nodes based on the content (e.g., data object creation time), context (e.g., node mobility as in most-social-first) or both (e.g., last node to see the data object's destination first, or minimize delay). Hui et al [26] identify the importance of using network centrality and social community knowledge when routing in packet switched networks, and propose a method that exploits both criteria in their technique called BUBBLE. Similar to this work, SOCRATIC estimates a node's centrality using a metric that encompasses the node degree over time. Unlike BUBBLE, SOCRATIC is designed for an ICN and uses the explicit subscriptions to infer social community, and includes link duration in the estimate of network centrality.

The discussed DTN approaches are agnostic of a data object's interests and are sender driven. Our research investigates DTN and ICN, which is receiver driven and has separate interest dissemination and content dissemination phases. SocialCast [16] is a utility-based replication scheme for DTN pub-sub systems that uses interests and a social heuristic to compute a single value for the utility of storing a replica at a particular node, using utility forecasting. In this scheme, data objects are forwarded only to nodes with a greater utility. The DIRECT protocol [40] replicates data objects towards the direction of the interest originator. Interests are propagated epidemically across connected components, and data objects traverse the reverse path of the interest propagation. Periodic purging and refreshing of interests ensure that stale paths are discarded and that new paths are discovered.

Several ICN architectures have been proposed as alternatives to the existing Internet architecture, including NDN [28], PURSUIT [42], NetInf [17], and MobilityFirst [39]. In the context of ICDTN, all of these approaches are limited due to their use of exact match naming, global name resolution services or traditional routing, or limited push-based information dissemination primitives [44]. SOCRATIC addresses all of these limitations; it generalizes exact match through a satisfaction degree metric, and uses this metric to push content which is subsequently routed using a disruption-tolerant protocol that does not require global connectivity.

Recent work [47] has demonstrated the value of combining network coding, ICN [35] and DTN [32], using interest-based routing. In the ICEMAN system, network coding can achieve significant performance benefits over fragmentation [29]. As an extension of ICEMAN, SOCRATIC uses DIRECT on network coded blocks when a path to the subscriber exists. Unlike existing approaches, SOCRATIC sets a global bound on the number of network encoded blocks for each data object, and a dynamic bound for each $\langle$ relay node, data object $\rangle$ pair. To the best of our knowledge, SOCRATIC is the first proposal to set this dynamic bound based on an interest satisfaction metric. Furthermore, none of the existing social ICN and DTN replication approaches operate on network encoded blocks.

3 Method

The beginning of wisdom is the definition of terms.

— Socrates

The approach to naming in SOCRATIC fits tactical networks very well since it eliminates the need for a global DNS by pushing content discovery into the bundle layer. In ICEMAN, only data objects that match an interest with a satisfaction greater than a threshold are transferred. Interest predicates for node S are represented as a set of weighted attribute/value pairs, $I(S) \subseteq \mathbb{A} \times \mathbb{V} \times \mathbb{N}$, where $\mathbb{A}$, $\mathbb{V}$, and $\mathbb{N}$ denote the domains for attributes, values, and weights. We say that content C with weighted attributes $M(C)$ satisfies interest $I(S)$ for node S with a satisfaction degree $\lambda(C,S)$ defined as in [47], where

$$\lambda(C,S) := \frac{\sum\{w_i \mid (a_i, v_i, w_i) \in I(S) \cap M(C) \times \mathbb{N}\}}{\sum\{w_i \mid (a_i, v_i, w_i) \in I(S)\}}. \tag{2}$$

We say that C matches S with threshold s, written as $C \models_s I(S)$, iff $\lambda(C,S) \geq s$. In other words, the normalized weighted sum of overlapping attributes between content $M(C)$ and interest $I(S)$ determines the satisfaction degree.

Interests are propagated epidemically and periodically purged. Content dissemination occurs either when an interest from node S with threshold s arrives at node P with a content C such that $C \models_s I(S)$, or when a data object C arrives at node R who has recently received an interest from node S such that $C \models_s I(S)$. In both cases we configured ICEMAN to replicate C to the neighbor from which P or R received the interests from S, effectively performing DIRECT.

After a data object C was published at node P and an interest from S has arrived such that $C \models_s I(S)$, P will generate network coded blocks $C_0, C_1, \ldots C_k$ to replicate towards S via relay R until either: 1) R and P become disconnected, or 2) R has received enough blocks to reconstruct C. We have limited the number of encoded blocks that the publisher can generate and transmit in order to avoid flooding the network with the content. Each block C_i is cached and routed just like any other data object, and inherits the attributes of C: $M(C_i) \supset M(C)$.

3.1 Popularity Index

SOCRATIC expands on ICEMAN as follows. To measure a node's popularity, each node R maintains a log of all of its previous contacts within a window of the past w_t seconds. This log is used to compute the node's average connect time A^R_{vt} and the number of encountered distinct neighbors N^R_e. Since SOCRATIC is deployed in a managed network, we assume that each node has an estimate of the network size,

N. We use the average connect time to compute a scalar N^R_{avt} given by

$$N^R_{avt} := \begin{cases} 2 & \text{if } A^R_{vt} \geq \beta_2, \\ 1 & \text{if } \beta_1 \leq A^R_{vt} < \beta_2, \\ 0 & \text{if } A^R_{vt} < \beta_1, \end{cases} \tag{3}$$

where β_1, β_2 are parameters that are set *a priori* according to network characteristics (15, 30 seconds in our scenarios). Node R computes its popularity index as

$$P^R := \frac{N^R_e * N^R_{avt}}{N}. \tag{4}$$

Nodes exchange their popularity indices during neighbor discovery. When content dissemination occurs at node P for content C and interest from neighbor R, if content $C \not\models_s I(R)$ (content C does not pass R's satisfaction threshold s) then the SOCRATIC algorithm is activated to compute the number of network coded blocks to generate and forward to the neighbor. If $C \models_s I(R)$, then there is a content match and P reverts to the previously described ICEMAN dissemination mechanisms (i.e., DIRECT). With SOCRATIC, P computes a satisfaction scalar $M(C, R)$ as

$$M(C,R) := \begin{cases} 2 & \text{if } 0.75 * s \leq \lambda(C,R) < s, \\ 1.5 & \text{if } 0.5 * s \leq \lambda(C,R) < 0.75 * s, \\ 1 & \text{if } 0.25 * s \leq \lambda(C,R) < 0.5 * s, \\ 0.5 & \text{if } \lambda(C,R) < 0.25 * s. \end{cases} \tag{5}$$

Let $B(C)$ be the number of blocks needed to decode C. P computes a maximum number of blocks of C to replicate to R, $B(C, R)$, as

$$B(C,R) := P^R * M(C,R) * B(C). \tag{6}$$

Let $\delta_P(C, R)$ be the number of generated blocks for C that are successfully sent by publisher P to R, and $\delta_P(C)$ be the total number of successfully sent blocks for C.

Each time P encounters some neighbor R, P greedily sends blocks to R to maximize $\delta_P(C, R)$ according to two constraints, namely,

$$C1)\ \delta_P(C, R) \leq B(C, R), \tag{7}$$

and

$$C2)\ \delta_P(C) \leq \gamma * B(C). \tag{8}$$

γ is a replication parameter that is set *a priori* according to network characteristics (1.5 in our scenarios).

3.2 Example Scenario

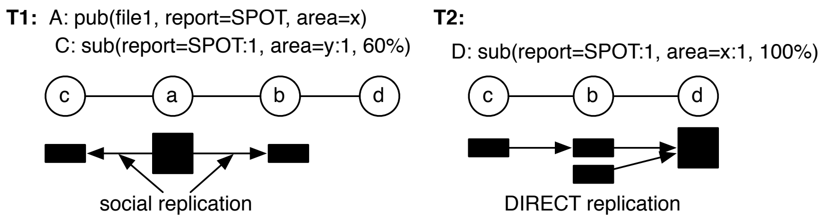
T1: A: pub(file1, report=SPOT, area=x)
C: sub(report=SPOT:1, area=y:1, 60%)
c
a
b
d
social replication
T2:
D: sub(report=SPOT:1, area=x:1, 100%)
c
b
d
DIRECT replication

Figure 4: An example scenario illustrating all of the mechanisms of SOCRATIC. The black square represents a decoded data object, and the rectangles represent encoded blocks of that data object.

Figure 4 illustrates a simplified scenario that demonstrates all of the mechanisms of SOCRATIC. At time T1 node a publishes *file1* with 2 attributes and node c subscribes to 2 attributes, each with a weight of 1, and a threshold of 60%, or $s = 0.6$. We assume $\gamma = 1.5$, $B(file1) = 4$, node b encounters a number of neighbors for durations such that $N^b_{avt} = 2$, $N^b_e = 2$, $P^b = 1$, and node c has $N^c_{avt} = 1$, $N^c_e = 1$, $P^c = 0.25$. Given the subscriptions, we have $M(file1, b) = 0.5$ and $M(file1, c) = 2$. Thus a computes $B(file1, b) = 1 * 0.5 * 4 = 2$ and $B(file1, c) = 0.25 * 2 * 4 = 2$, and will generate 4 linearly independent encoded blocks: 2 that are sent to c and 2 that are sent to b by constraint C1 (constraint C2 does not apply since the number of blocks, 2, is less than $1.5 * 4 = 6$). Later, at time T2, c and b are connected with d which has an exact match interest with the original data object, and thus its encoded blocks. Since there is a stable path between c,b,d and $file1 \models_1 I(d)$, DIRECT is triggered which replicates these blocks to d who subsequently reconstructs the file. Note that decoding only occurs at the subscriber, and that the publisher is solely responsible for determining the number of blocks generated.

3.3 Discussion

From these definitions and the example scenario, it is clear that the number of encoded blocks that are generated for a neighbor is proportional to a node's popularity and its interest commonality with the data object. N^R_{avt} acts as a filter to prevent sending blocks to nodes with a very short contact time despite their popularity (i.e. a lossy interface), and gives more weight to nodes that have a longer contact time. Through N^R_e, nodes that meet many other nodes will receive more blocks. $M(C, R)$ exploits the observation that in tactical networks nodes with similar interests tend to be co-located. Replication limitations do not occur if the neighbor has subscriptions that match the data object, or if the neighbor is along a stable route to a subscribed node. Regular ICEMAN is structured so that either data objects do or do not match an interest, however SOCRATIC uses the satisfaction degree directly to make decisions. Only the publisher controls the number of blocks that are generated in the network for a data object, and places these blocks carefully. Since the blocks are treated as any other data object, they are subsequently routed to subscribers if a stable path exists.

From a theoretical perspective, if D denotes the number of published data objects at a particular node, and N_e denotes the number of encountered neighbors, then this algorithm requires $\mathrm{O}\big(D * N_e\big)$ additional memory at the node when compared to ICEMAN. The other additional memory costs are negligible. In practice, the block counters and data objects should be aged and purged to save memory. Typically data object purging is done through a maximum hop-count TTL or timestamp which indicates when the content is no longer useful, although more complicated policies are supported such as enforcing a total order on classes of content (e.g., video frame data objects with a sequence number attribute). These policies are enforced by the utility caching framework as described in Section 1.5. The parameters w_t, β_1, β_2,

γ are network dependent and are determined *a priori*, typically through extensive emulation.

4 Experiments

Two important characteristics of maps should be noticed. A map is not the territory it represents, but, if correct, it has a similar structure to the territory, which accounts for its usefulness.

— Alfred Korzybski

We evaluated the implementation using the Automatic Network Test System (ANTS, described in Section 4.1) with the CORE [8] network emulator, where each node is associated with a Linux light-weight container running SOCRATIC. We used the CORE basic MAC which does not model layer 1 or 2 phenomena in detail (lossless 11Mbps pipes with 50ms delay). Each node has a broadcast radius of 250m and unlimited cache capacity. Each data object is 1MB and the block size is 32KB. We compared SOCRATIC against four other content dissemination algorithms with network coding enabled: DIRECT[3], EPIDEMIC, SPRAY, and sSPRAY. DIRECT is ICEMAN as previously described, with DIRECT routing and constraint C2 (defined in Section 3.1). EPIDEMIC is ICEMAN with Epidemic [45] routing and constraint C2. SPRAY is SOCRATIC without DIRECT routing and only constraint C2 (it replicates on first encounter). sSPRAY (social SPRAY) is SOCRATIC without DIRECT. The SPRAY approach is similar to social oblivious replication approaches that replicate on first encounter, while sSPRAY incorporates the social popularity index to give replication preference to more social nodes. All experiments are averaged over 4 runs, each with a different mobility seed.

4.1 Automatic Network Test System (ANTS)

In this section, we describe ANTS (Automatic Network Test System), a network emulator harness that was designed to evaluate SOCRATIC. It increases test reproducibility and ease of development for networks system evaluation that require a detailed model. We first motivate the design of this evaluation system, and then

[3] The interest refresh was set to 20 seconds, with a 30 second purge.

discuss the technical details of its implementation. Although a full description is outside the scope of this thesis, we detail the system here to elucidate the evaluation methodology for SOCRATIC.

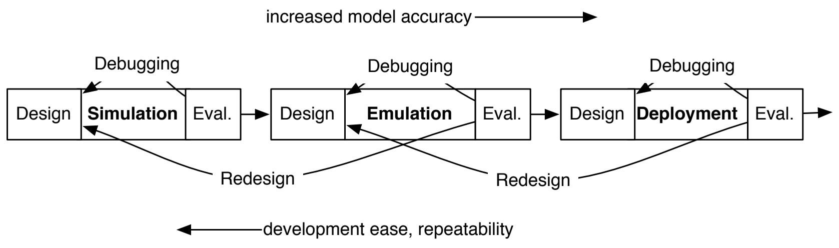

Figure 5: The design, debug, and evaluate process cycle that was used when developing ICEMAN and SOCRATIC.

We found that designing and implementing a new network architecture requires multiple iterations of development and evaluation using models at different levels of detail, for both the architecture implementation and the targeted network environment. For example, the routing and network coding modules of ICEMAN and SOCRATIC were first implemented as software within QualNet [7], a discrete-time event simulator which provides an API with a high ease of development and whose experiments generate highly repeatable results. After undergoing multiple iterations of development, debugging and and evaluation at the simulator stage, the protocols needed to be implemented together on a single stack, and evaluated on a network with a more detailed model, in order to understand how the system would behave in a disaster response or tactical scenario with current hardware. The implementation at the emulation stage used the full network stack and was very close to the end solution that ran on mobile phones. In the context of ICEMAN, we then deployed the implementation on actual hardware and evaluated the system using only network emulation. Lastly, we performed over-the-air (OTA) tests in a target network. Debugging occurred during the design and evaluation of the architecture at each stage, but major problems with a module required it to be re-designed and evaluated at a previous stage. Fig. 5 illustrates this development approach, and the spectrum of trade-offs for development ease, repeatability, and model accuracy.

Discrete-time simulators [6, 7] allow the developer to write their protocols at a high level (by abstracting the network API) and specify network parameters without worrying about the actual hardware or specific network characteristics. Multiple instantiations of the protocol pass messages to each other via a FIFO queue which adds delays, jitter (variance in delay), and errors that are typical of a target network deployment. The protocol specification is abstract enough that time can be virtualized and each message is immediately processed in a fast simulation time. The simulation stage allows the designer to quickly understand at a high-level how an idealized implementation will perform in an idealized network. The designer can rapidly prototype different versions of the protocol and quickly discover sources of protocol design errors and scalability issues.

The software implementation at the simulation stage is simple and does not need to handle concurrency, hardware specific driver code, or use a real network socket API which may have operating system dependent idiosyncrasies. These abstractions in the implementation model drastically reduce the development time by eliminating the need to write difficult to debug code. The network model is also abstracted, and may not run the full targeted network stack (each network layer runs its own abstracted code). This network abstraction enables repeatability, where each test case specifies a unique seed to ensure the same randomness (e.g. jitter, packet loss, mobility) occurs across multiple executions of the same test. Due to the simplified model, simulators typically provide an easy way to generate a multitude of repeatable tests. The disadvantage of simulation is that it does not capture low level phenomena which may dramatically effect the end system performance and system implementation difficulty. For example, SOCRATIC uses network coding which requires matrix inversion calculations and an advanced level of code sophistication to achieve correctness and performance. First, the network coding functionality was evaluated within QualNet to verify correctness and to initially understand the benefits of the approach [32]. However, since QualNet does not model CPU latency, it was necessary to then evaluate the network coding component within SOCRATIC in emulation in order to understand how feasible the approach was in practice. Could network coding even be deployed on mobile phones with limited resources? The answer is yes, but the parameters are important [29]. The results of emulation uncovered the tradeoffs between fragmentation size, rate, and performance.

Network emulation [8, 11] executes instances of the software implementation either as separate processes, light weight containers, or virtual machines, and models the network communication between these instances in real time. Since the network is emulated, the designer can debug an implementation that is close to the final implementation on a variety of network topologies, whereas actual deployment is typically time and cost prohibitive when still at the design stage. The implementation at this stage is mostly complete, but is missing hardware specific driver code if the targeted hardware is not emulated. The network emulation is more detailed than that of a simulator, and the actual network stack may be used while only the physical layer is abstracted (e.g., using a pipe or a socket). The test reproducibility is reduced due to the use of random number generators internal to the emulator, and due to timer dependent code: the same test cases may give different results depending on the hardware and operating system that executes the emulation. Indeed, we found that in the early stages of our research machines with a faster CPU had better evaluation results than machines with a slower CPU, an indication that it would be difficult to make quantitative claims that were reproducible. Typically the emulated networks are very specific and the emulation tools do not provide a way of generating many test cases, as in a simulator. In practice, creating multiple runs with different parameters must be done manually, which is time consuming and error prone. In comparison to simulation, errors in parameter assignment often take a long time to discover, since they appear at the end of test execution and emulation runs in real-time. What is needed is a system for network evaluation that can achieve the high model detail of emulation with the repeatability and ease of a simulator. We designed ANTS in response to this need. ANTS is a test harness for a network emulator (currently CORE) that enables the designer to succinctly specify tests which are compiled to fully automated test cases that can be executed with greater ease and reproducibility than what is available with traditional network emulation.

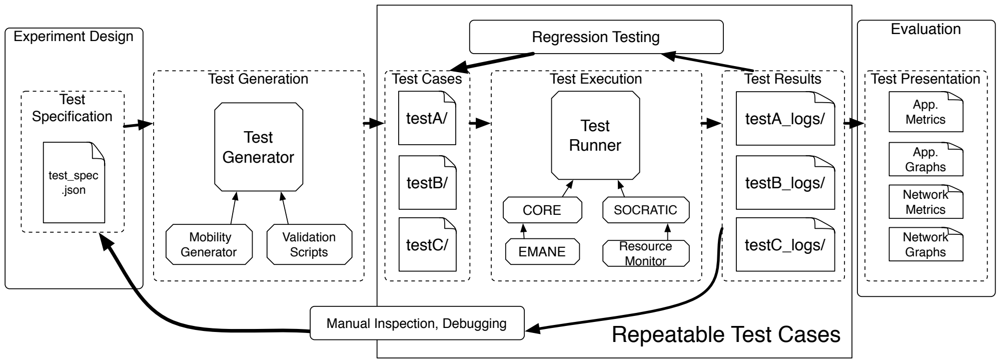

Figure 6: An architecture diagram and work flow of the Automatic Network Test System that was used to evaluate SOCRATIC in a realistic, yet repeatable way.

Figure 6 illustrates the ANTS architecture and work flow of test generation and evaluation. A user creates first creates a test specification file that is in JavaScript Object Notation (JSON) format that defines the entire network and application parameters, such as the number of nodes, duration, mobility, and traffic parameters. We use JSON since it is human readable, easy to modify, and it lends itself to Web services should the framework have a Web frontend. Figure 7 shows the top portion of a test specification file to illustrate how the parameters are defined and the use of the repeat and assign metacommands (prefixed with the ˆsymbol and within a dictionary). This specification defines a test that has 22 nodes, a duration of 1880 seconds, and covers a $1300x1300m^2$ area.

The repeat metacommand is used to automatically generate a number of test cases that are identical, except for a few parameters which vary. The `^repeat` keyword takes an array as a value, and will create a test case for every element in the array, where the key for which `^repeat` is a value is assigned each element. For example, in Figure 7 the `seed` is assigned 1 through 5 in different test cases. The metacommands can compose, to create every permutation for the repeat elements. To see this functionality, in Figure 7 the `config` key has a `^repeat` command and gets assigned dictionaries with keys `^assign` and `%%diss%%`. When combined with the `seed` repeat, this test specification generates a total of 10 test cases, one for every combination of `seed` and `config` assignments. This metacommand eases the generation of a large suite of test cases, and reduces human errors by allowing the user to only specifying what parameters to vary.

The assign metacommand is used for string replacement. The `^assign` keyword takes the value and assigns it as the value for which the metacommand dictionary (the enclosing dictionary { } of the metacommand) is a value. Any other keys specified in the metacommand's enclosing dictionary are then used to string replace all occurrences of that key with the key's value. For example, in Figure 7 the string `%%diss%%` will be replaced with `socratic` or `epidemic`, depending on the assignment from repeat. In this way, each test case that uses the `config-socratic.xml` configuration file will have the `output` set to `experiment-socratic`, and each test cases that uses the `config-epidemic.xml` configuration file will have the `output` set to `experiment-epidemic`. The `name` is used to specify an output directory when running experiments, thus the experiment results for SOCRATIC and EPIDEMIC will be stored in separate directories. This metacommand is invaluable when creating a test specification that results in dozens if not hundreds of test cases, as one can easily find the key parameters for each test output by reading the directory name.

The specification may also reference external files such as application configuration (e.g., the `config-socratic.xml` and `config-epidemic.xml` lines in Figure 7), validation scripts or traffic generation scripts which will be executed. This specification and the external files are passed to the *Test Generator* which acts as a compiler that parses the JSON and outputs multiple stand-alone test cases which are distributed separately from the Test Generator, and are executed using the *Test Runner*. A test case is a bundle that contains all of the dependent scripts and libraries, so that it can be run on any environment that supports the Test Runner. The Test Generator uses mobility model libraries (e.g., BonnMotion [1] for Random Waypoint and Nomadic mobility) when generating tests, and these models and their parameters are included in the test specification and automatically passed to these tools. The user can optionally add validation scripts which are executed at the end of the test to verify that the applications behaved correctly and that the test passed. The decision to separate test generation from test execution was deliberate. This demarcation enables the Test Generator and Test Runner to be developed independently, and relieves the evaluator from searching for the test generator and its dependencies to recompile the tests. This separation is analogous to the tools and stages to C development: C source code (i.e., test specification), a compiler (i.e., test generator), its binary object code (i.e., test case), and an execution environment (i.e., test runner). Binaries (i.e., test cases) are easier to distribute and execute, but changes require the source code (i.e., the test specification).

The Test Runner loads test cases by reading a file which specifies the paths to the test cases, along with the number of times to execute each test. The Test Runner will initialize the emulator and perform all the initialization necessary to start the architecture and the applications. In Figure 6 we see that the Test Runner initializes the CORE emulator for network emulation, with EMANE [4] for realistic physical layer modeling, SOCRATIC (the application under test), and a Resource Monitor. The Resource Monitor keeps track of system resource utilization to ensure that the system under test does not run out of resource and generate invalid results. A separate process periodically polls the system for data (i.e., CPU, memory, and file descriptor usage) and stores this information in a log that is checked by the Test Runner at the end of the test execution to ensure that no resource limits were violated during the test. The test specification can set a maximum CPU usage for each light weight container, using the Linux program cpulimit [2], to enforce container fairness or to model a CPU constrained device.[4] An optional daemon can be activated which will periodically send a broadcast ping to each neighbor and record replies—this connectivity data is used to build network connectivity graphs.

At the end of the test execution (the exact duration is in the specification) the Test Runner ensures that all of the emulation and application processes have exited and kills any leftover processes so that the system has a clean slate for the execution of the next test. The Test Runner collects all of the log files generated by the emulator, applications, and resource monitor, and stores them in a single output directory along with the test specification file. To increase repeatability, the Test

[4] We found that cpulimit was incompatible with some applications, since it bounds CPU usage using the SIGSTOP and SIGCONT POSIX signals which can cause other interrupts to be lost.

Runner generates a file in the output directory that contains system information such as the version of Linux, hardware details, and all of the installed packages. The output files are then amenable to graph generation and metric aggregation.

As illustrated by the thick arrows in Figure 6, ANTS has 3 main work flows: 1) development and debugging, 2) regression testing, and 3) evaluation. All 3 of these work flows were used in the development of SOCRATIC. In work flow 1) a developer generates a series of tests, executes the tests, examines the output, and uncovers bugs in the protocol that could manifest as crashes, poor performance or error messages. They then fix the bug, update the code, and re-run the tests. In work flow 2) a developer generates a series of test cases with validation scripts that ensure that the system behaves in a predetermined way. Usually these tests exercise bare-bones functionality or test features in ways that were previously broken. Regression tests are executed whenever a new version of the system is created to ensure that previously existing functionality still works in the new version. Lastly, work flow 3) occurs once the system has become stable and requires evaluation. In this work flow, a series of automated scripts parse the output logs for the application and the network to generate graphs and aggregate metrics. Using ANTS in these 3 ways greatly reduced the development time of SOCRATIC by quickly uncovering bugs and sources of protocol inefficiencies, and by eliminating a class of human errors where experiments were conducted with incomparable parameters.

4.2 Disconnected Scenario

The disconnected scenario models 4 squads of 5 nodes on patrol in 4 different regions, with 2 UAVs that circle between the squads. The UAVs cannot communicate directly with each other. Figure 8 illustrates the emulated scenario and Table 1 summarize the parameters. The total area for the emulation is 1300x1300 m^2 and the duration of each experiment is 30 minutes. There are two UAVs flying around the area with a velocity of 30 $meters/sec$ around a circle with radius of 550 meters. Each member of a squad has a mobility pattern of Random Waypoint with minimum and maximum velocity of 1 and 7 $meters/sec$, respectively. The minimum and maximum pause test_spec.json

```
[
{
    "num_nodes"          : 22,
    "duration"           : 1880,
    "seed"               : { "^repeat" : [1, 2, 3, 4, 5] },
    "area_height"        : 1300,
    "area_width"         : 1300,
    "config"             : {"^repeat" : [
        { "^assign" :
            {
                "path" : "./Configs/config-socratic.xml"
            },
          "%%diss%%" : "socratic"
        },
        { "^assign" :
            {
                "path" : "./Configs/config-epidemic.xml"
            },
          "%%diss%%" : "epidemic"
        }
    ],
    "output" : "experiment-%%diss%%",
...
    },
...
```

Figure 7: An example portion of a test specification file. It is in the JSON format with metacommands `^repeat` and `^assign` that the Test Generator uses to produce multiple test files that are executed by the Test Runner.

time for each group member is zero and five seconds, respectively. 25 data objects are published 5 minutes into the test, where each data object has a single subscriber with a 100% satisfaction degree and a 100% threshold. The squad member subscriptions occur 15 minutes into the test, and the UAV subscriptions occur at the beginning of the test. The UAVs have 75% satisfaction degree with all of the data objects (with a 100% threshold). Publisher-subscriber pairs are chosen uniformly at random, with the constraint that the publisher and subscriber belong to different squads. This constraint implies that all inter-squad communication must occur through the UAV message ferries.

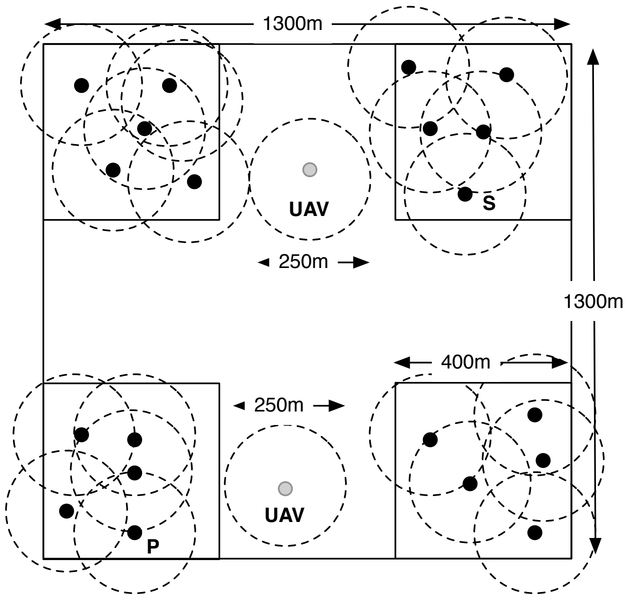

Figure 8: Disconnected scenario. Node "P" publishes a data object that only node "S" subscribes to. "UAV" nodes ferry the data object from "P" to "S".

Total Area	1300x1300m^2
Total Nodes	22
Duration	30 minutes
UAV Mobility	Circle (550m radius)
UAV Velocity	30m/s
Squad Member Mobility	Random Waypoint
Squad Member Min, Max Velocity	1,7m/s
Squad Member Min, Max Pause Time	0,5s

Table 1: Disconnected scenario parameters.

5 Results

Figure 9 graphs the delivery versus the delay across the different dissemination approaches. Since Epidemic dissemination is usually considered a baseline for DTN routing protocols, our comparison shows that SOCRATIC outperforms this approach and delivers the most data objects within the delay constraint. It is only when the delay constraint is very high that EPIDEMIC slightly outperforms SOCRATIC because the network is less congested after the initial flood of data objects which results in faster delivery while SOCRATIC only caches the data object judiciously to save network resources. The figure also shows that SOCRATIC clearly outperforms other techniques for all level of delay constraints.

In comparing SPRAY and sSPRAY, it is clear that the popularity index captures the higher social value of the UAVs, enabling sSPRAY to replicate more blocks directly to the UAVs than SPRAY which replicates most of the blocks to its immediate neighbors who are not in contact with the subscriber. This comparison highlights how social information is used to discover better relays. In comparing DIRECT and sSPRAY, we see the benefit of replicating blocks multiple hops along the path that the last interest was received—this path tends to be through a UAV directly or an intermediate node to a UAV if it recently received the interest. In comparing DIRECT and SOCRATIC, we see how placing replicas closer to the destination prior to routing consistently reduces delay.

Figure 10 shows the total bandwidth utilized (for transmit (Tx) and received (Rx)) along with total data object (DO) delivery. SOCRATIC delivers 100% of data objects while other techniques deliver less that 100% except EPIDEMIC approach. This reduction in bandwidth usage enables SOCRATIC to deliver most of the data objects within less delay than using EPIDEMIC which quickly saturates the bandwidth. Moreover, SOCRATIC disseminates the information intelligently and therefore requires less bandwidth usage for the same delivery compared to EPIDEMIC approach. Given the interference and scarcity of spectrum in dense wireless environments, SOCRATIC clearly outperforms all of these techniques by caching selectively.

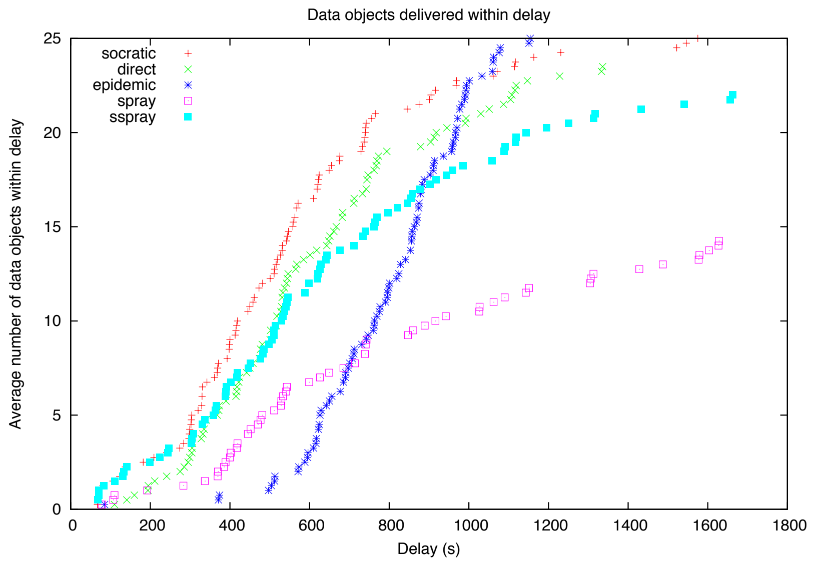

Figure 9: Average delay distribution in the disconnected scenario for the different representative replication approaches.

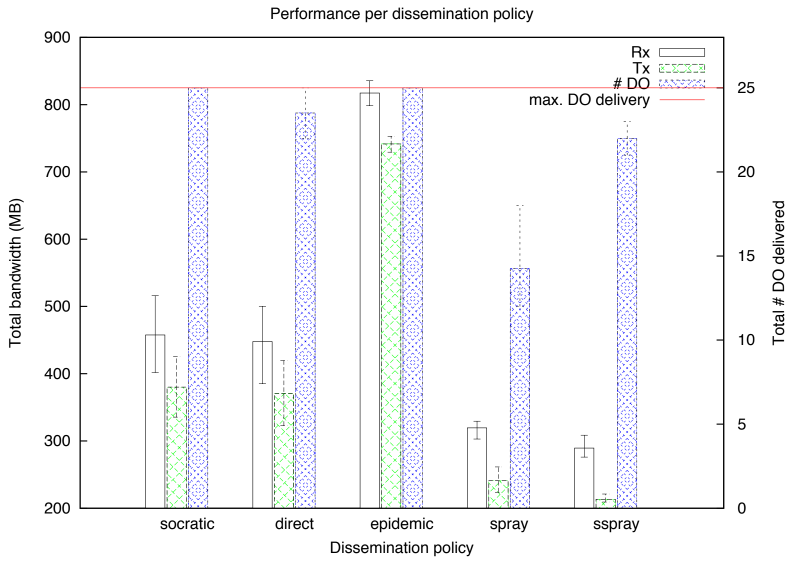

Figure 10: Average transmit megabytes (Tx), receive megabytes (Rx), and data objects delivered (#DO) for the disconnected scenario. The error bars represent the min and max across all runs. The red line indicates the maximum possible number of data objects delivered (max. DO delivery).

6 Discussion

Research into efficient dissemination in ICDTN led to the design of a novel replication approach called SOCRATIC that unifies replication and network coding to efficiently disseminate content through delay-tolerant networks (DTNs). The main idea behind SOCRATIC is to cache network encoded blocks in nodes that are more relevant in terms of their interest in the original data object, and nodes that have a higher probability of meeting another node that has interest in this content by computing a popularity index. The popularity index computes a value that is proportional to the interest commonality of node to the data object, average contact time for the node and the average number of nodes that the node meets. SOCRATIC takes advantage of the distributed nature of random linear network coding [24] and controls the number of encoded blocks that a cache receives. This judicious distribution of the encoded blocks allows the network to utilize minimum network resources (bandwidth) while maximizing the data delivery. Emulation results have shown that this technique outperforms several replication approaches proposed in the DTN literature.

Future work should examine the role the choice in parameters plays in influencing performance. Can the parameters be discovered dynamically through a learning phase, or can a portfolio of parameters be used that is selected based on mission context (e.g., similar to adaptive interest modeling in [33])? This work can be integrated with a more general utility-based dissemination framework, and can apply different replication policies according to content priority. Actual publish, subscription, and mobility traces could be used in an evaluation to determine how well SOCRATIC captures the social structure in deployed ICDTN applications. ANTS (Section 4.1) could be expanded to evaluate SOCRATIC on an automated mobile test bed using an inexpensive hardware platform, such as quad-copters and the iRobot Create [5]. This work can be extended to wireless mobile ad hoc networks (MANETs) where nodes are mainly connected in the network. In such networks, the popularity index can be redefined to consider the unique characteristics of MANETs.

References

[1] BonnMotion - a mobility scenario generation and analysis tool. `http://sys.cs.uos.de/bonnmotion/`. Accessed: 2014-08-01.

[2] cpulimit. `https://github.com/opsengine/cpulimit`. Accessed: 2014-08-01.

[3] Creating a secure, private internet and cloud at the tactical edge. `http://www.darpa.mil/NewsEvents/Releases/2013/08/21.aspx`. Accessed: 2014-07-18.

[4] Extendable mobile ad-hoc network emulator. `http://www.nrl.navy.mil/itd/ncs/products/emane`. Accessed: 2014-08-01.

[5] iRobot create programmable robot. `http://en.wikipedia.org/wiki/iRobot_Create`. Accessed: 2014-08-06.

[6] NS-3. `http://www.nsnam.org`. Accessed: 2014-07-29.

[7] SCALABLE network technologies, QualNet. `http://web.scalable-networks.com/content/qualnet`. Accessed: 2014-07-29.

[8] Jeff Ahrenholz. Comparison of CORE network emulation platforms. In *MILCOM*, pages 166–171. IEEE, 2010.

[9] Aruna Balasubramanian, Brian Levine, and Arun Venkataramani. DTN routing as a resource allocation problem. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 373–384, 2007.

[10] Burton Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM 13*, 1970.

[11] Marta Carbone and Luigi Rizzo. Dummynet revisited. In *ACM SIGCOMM Computer Communication Review*, 2010.

[12] Antonio Carzaniga and Alexander L. Wolf. Content-based networking: A new communication infrastructure. In *Revised Papers from the NSF Workshop on Developing an Infrastructure for Mobile and Wireless Systems*, 2002.

[13] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss. RFC 4838. *IETF*, 2007.

[14] Narottam Chand, Ramesh C Joshi, and Manoj Misra. Cooperative caching in mobile ad hoc networks based on data utility. In *Mobile Information Systems*, 2007.

[15] William Cook, William Cunningham, William Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. Wiley-Interscience, 1997.

[16] Paolo Costa, Cecilia Mascolo, Mirco Musolesi, and Gian Pietro Picco. Socially-aware routing for publish-subscribe in delay-tolerant mobile ad hoc networks. *Journal on Selected Areas in Communications*, 26(5):748–760, 2008.

[17] Christian Dannewitz, Dirk Kutscher, Börje Ohlman, Stephen Farrell, Bengt Ahlgren, and Holger Karl. Network of information (NetInf)–an information-centric networking architecture. *Computer Communications*, 2013.

[18] Paul Erdős and Alfréd Rényi. On a classical problem of probability theory. *Magyar Tudomos Akada Matematikai Kutattk Kzlemei*, 1961.

[19] William Finn. Improving battlefield connectivity for dismounted forces. *Defense Tech Briefs*, 2012.

[20] Christina Fragouli, Jean-Yves Le Boudec, and Jörg Widmer. Network coding: an instant primer. *ACM SIGCOMM Computer Communication Review*, 2006.

[21] Skyler Frink. Secure cell phone technology gets ready for deployment. *Military and Aerospace Electronics*, 2012.

[22] Ali Ghodsi, Teemu Koponen, Jarno Rajahalme, Pasi Sarolahti, and Scott Shenker. Naming in content-oriented architectures. *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, 2011.

[23] Ali Ghodsi, Scott Shenker, Teemu Koponen, Ankit Singla, Barath Raghavan, and James Wilcox. Information-centric networking: seeing the forest for the trees. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, page 1. ACM, 2011.

[24] Tracey Ho, Ralf Koetter, Muriel Medard, David R Karger, and Michelle Effros. The benefits of coding over routing in a randomized setting. In *International Symposium on Information Theory (ISIT)*, page 442, 2003.

[25] Pan Hui, Augustin Chaintreau, James Scott, Richard Gass, Jon Crowcroft, and Christophe Diot. Pocket switched networks and human mobility in conference environments. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, 2005.

[26] Pan Hui, Jon Crowcroft, and Eiko Yoneki. Bubble rap: Social-based forwarding in delay-tolerant networks. *Transactions on Mobile Computing*, 10(11):1576–1589, 2011.

[27] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, 2000.

[28] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, 2009.

[29] Joshua Joy, Yu-Ting Yu, Mario Gerla, Samuel Wood, James Mathewson, and Mark-Oliver Stehr. Network coding for content-based intermittently connected emergency networks. In *Proceedings of the 19th Annual International Conference on Mobile computing and Networking*, pages 123–126. ACM, 2013.

[30] Vikas Kawadia, Niky Riga, Jeff Opper, and Dhananjay Sampath. Slinky: An adaptive protocol for content access in disruption-tolerant ad hoc networks.

In *ACM MobiHoc 2011 International Workshop on Tactical Mobile Ad Hoc Networking*, 2011.

[31] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. In *ACM SIGCOMM Computer Communication Review*, 2007.

[32] Uichin Lee, Joon-Sang Park, Joseph Yeh, Giovanni Pau, and Mario Gerla. Code torrent: content distribution using network coding in vanet. In *Proceedings of the 1st International Workshop on Decentralized Resource Sharing in Mobile Computing and Networking*, pages 1–5. ACM, 2006.

[33] Hua Li, Ralph Costantini, David Anhalt, Rafael Alonso, Mark-Oliver Stehr, Carolyn Talcott, Minyoung Kim, Timothy McCarthy, and Samuel Wood. Adaptive interest modeling enables proactive content services at the network edge. *UMAP Project Synergy (ProS)*, 2014.

[34] Anders Lindgren, Avri Doria, and Olov Schelén. Probabilistic routing in intermittently connected networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 7(3):19–20, 2003.

[35] Marie-Jose Montpetit, Cedric Westphal, and Dirk Trossen. Network coding meets information-centric networking. In *Proceedings of the 1st ACM workshop on Emerging Name-Oriented Mobile Networking Design-Architecture, Algorithms, and Applications*, 2012.

[36] Erik Nordström, Per Gunningberg, and Christian Rohner. A search-based network architecture for mobile devices. Technical report, Department of Information Technology, Uppsala University, 2009.

[37] Soon-Young Oh, Davide Lau, and Mario Gerla. Content centric networking in tactical and emergency MANETs. In *Wireless Days, IFIP*, pages 1–5. IEEE, 2010.

[38] Konstantinos Psounis and Cauligi S Raghavendra. Spray and wait: An efficient routing scheme for intermittently connected mobile networks. In *Proceedings of the 2005 ACM SIGCOMM Workshop on Delay-Tolerant Networking (WDTN)*, 2005.

[39] Dipankar Raychaudhuri, Kiran Nagaraja, and Arun Venkataramani. MobilityFirst: a robust and trustworthy mobility-centric architecture for the future Internet. *ACM SIGMOBILE Mobile Computing and Communications Review*, 2012.

[40] Ignacio Solis and JJ Garcia-Luna-Aceves. Robust content dissemination in disrupted environments. In *Proceedings of the Third ACM Workshop on Challenged Networks*, pages 3–10. ACM, 2008.

[41] Thrasyvoulos Spyropoulos, Thierry Turletti, and Katia Obraczka. Routing in delay-tolerant networks comprising heterogeneous node populations. *IEEE Transactions on Mobile Computing*, 8(8):1132–1147, 2009.

[42] Dirk Trossen and George Parisis. Designing and realizing an information-centric internet. *IEEE Communications Magazine*, 2012.

[43] Gareth Tyson, John Bigham, and Eliane Bodanese. Towards an information-centric delay-tolerant network. In *Proceedings of IEEE INFOCOM Workshop on Emerging Design Choices in Name-Oriented Networking (NOMEN)*, 2013.

[44] Gareth Tyson, Eliane Bodanese, John Bigham, and Andreas Mauthe. Beyond content delivery: Can ICNs help emergency scenarios? *IEEE Network*, 2014.

[45] Amin Vahdat and David Becker. Epidemic routing for partially connected ad hoc networks. Technical report, CS-200006, Duke University, 2000.

[46] Forrest Warthman. Delay-tolerant networks (DTNs) a tutorial. Technical report, Warthman Associates, 2003.

[47] Samuel Wood, James Mathewson, Joshua Joy, Mark-Oliver Stehr, Minyoung Kim, Ashish Gehani, Mario Gerla, Hamid Sadjadpour, and J.J. Garcia-Luna-

Aceves. ICEMAN: A system for efficient, robust and secure situational awareness at the network edge. In *IEEE MILCOM*, 2013.

[48] Wenrui Zhao, Yang Chen, Mostafa Ammar, Mark Corner, Brian Levine, and Ellen Zegura. Capacity enhancement using throwboxes in DTNs. In *Mobile Adhoc and Sensor Systems (MASS)*, pages 31–40. IEEE, 2006.