CBMEN ENCODERS

Content-Based Mobile Edge Networking Progam

Edge Networking with Content-Oriented Declarative Enhanced Routing and Storage

Contract Number: N66001-12-C-4051

Phase 2 ENCODERS Final Report

June 8, 2016

Submitted to:

Defense Advanced Research Projects Agency (*DARPA*), Strategic Technology Office (*STO*)
Attention: Dr. Wayne Phoel, Email: wayne.phoel@darpa.mil

Submitted by:

SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025 USA
Attention: Dr. David E. Wilkins, Phone: (650) 859-2057, Email: david.wilkins@sri.com

Distribution Statement A: Approved for Public Release, Distribution Unlimited.

Disclaimer: The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

Table of Contents

1 Scope 1
1.1 Program Objective 1
1.2 SRI's Team and Program Role 2
2 Overview of ENCODERS Design 4
2.1 Overall Architecture 4
2.2 ENCODERS Architecture Leveraging Haggle 5
2.2.1 Functional Details 7
2.2.2 Subtleties 8
3 Demonstration 9
4 Technical Solution and Accomplishments 13
4.1 Content-Based and Utility-Based Dissemination 14
4.1.1 Interest-Driven Content-Distribution Approach 14
4.1.2 A General Architecture for Content Distribution 15
4.1.3 Limitations and Future Work 17
4.2 Content-based Protocols: Lightweight Unicast and Broadcast 17
4.2.1 Lightweight Unicast and Broadcast via UDP 18
4.2.2 Limitations and Possible Future Directions 19
4.3 Content-Based and Utility-Based Caching 20
4.3.1 Content-Based Caching Approach 20
4.3.2 Utility-Based Caching Framework 23
4.3.3 Applications of Content-Based Caching 23
4.3.4 Data Store Optimization and Memory Usage Control 24
4.3.5 Limitations and Possible Future Directions 25
4.4 Network Coding and Fragmentation 25
4.4.1 Network-Coding Approach 26
4.4.2 Implementation of Network Coding and Fragmentation 26
4.4.3 Context-Aware Network Coding Implementation 28
4.4.4 Limitations and Possible Future Directions 29
4.5 Security 30
4.5.1 Overview 30
4.5.2 Approach 31
4.5.3 Using Access-Control Policies 34
4.5.4 Concept of Operations (CONOPS) for Security 35
5 Evaluation 38
5.1 Methodology and Robustness Results 39
5.2 Summary of Phase 1 Evaluation Results 46
5.3 Phase 2 Evaluation Results 46
5.3.1 Ground-based platoon-level operations 46
5.3.2 Extended Scenario 59
Bibliography 65

Table of Figures

Figure 1. ENCODERS capabilities within the broader CBMEN Program context. 4
Figure 2. ENCODERS Architecture and API. 6
Figure 3. Content delivery, latency, and availability from Government Field Test 9
Figure 4. The performance of ENCODERS cache purging options 10
Figure 5. The performance of network coding.....11
Figure 6. Our team's four Phase 1 sidebar demonstrations using ENCODERS.....12
Figure 7. Network Coding. 27
Figure 8. The security solutions developed in ENCODERS.....30
Figure 9. SRI's Test Automation Framework 40
Figure 10. Bandwidth consumption 41
Figure 11. Cumulative CPU usage.....42
Figure 12. Cumulative memory consumption.....42
Figure 13. Number of physical neighbors at any given time 43
Figure 14. Number of the logical neighbors at any given time 43
Figure 15. The percentage of cache capacity used by each node. 44
Figure 16. The eviction and insertion of data objects in the cache over time.....45
Figure 17. Data object delivery.....50
Figure 18. Cumulative data objects delivered, *static provisioning*.....51
Figure 19. Cumulative data objects delivered, *dynamic provisioning*.....52
Figure 20. Cumulative data objects delivered, no *security policy*.....52
Figure 21. Data object delivery for different policies, Android devices, 30 nodes 53
Figure 22. Cumulative data objects delivered, on Android devices. 54
Figure 23. Data object delivery for different policies, Linux containers, 50 nodes 55
Figure 24. Delivery performance in a 50-node network 56
Figure 25. Data object delivery, 30-node Linux containers, limited cache 57
Figure 26. A utility-based view of the effects of limiting cache size.....58
Figure 27. Extended scenario 60
Figure 28. Data-object delivery latency with and without satellite 61
Figure 29. Cumulative data objects delivered, with satellites.....63
Figure 30. Data object delivery over time.....64

1 Scope

SRI's ENCODERS (Edge Networking with Content-Oriented Declarative Enhanced Routing and Storage) software was developed for DARPA's *Content-Based Mobile Edge Networking (CBMEN)* Program. This document is the SRI Final Report for Phase 2 of the CBMEN Program (Contract Number: N66001-12-C-4051). The period of performance ends on 25 September 2014.

Section 2 describes SRI's design within the context of the overall CBMEN Program. Next, it describes the top-level design of ENCODERS, which is based on the Haggle open-source code base for content-based networking. Section 3 summarizes our field demonstration at the end of Phase 1. Section 4 describes the design of each major enhancement that SRI made, including the concept of operations (CONOPS) for using our design for security in the domain of small-unit operations of warfighters at the edge (see Subsection 4.5.4).

Evaluation of ENCODERS has a major task and Section 5 describes the metrics, methodology, and evaluation results for ENCODERS. In addition to this report, we delivered three other documents:

- ENCODERS Software Design Description (with a detailed section for the security CONOPS).
- ENCODERS Configuration Manual, which is intended for system administrators who are setting up an ENCODERS network.
- ENCODERS Technical Documentation, which is intended for developers who may be enhancing or modifying the system (e.g., it describes software classes and events).

The Haggle documentation at https://code.google.com/p/haggle/ completes the documentation suite for ENCODERS.

1.1 Program Objective

The objective of the DARPA's *Content-Based Mobile Edge Networking (CBMEN)* program is to develop the network services and transport architectures to enable efficient, transparent distribution of content in mobile ad hoc networking (MANET) environments. CBMEN envisions application-independent and network-agnostic content-distribution services that will be utilized by battlefield applications to efficiently distribute content. The goal is to reduce latency and increase the effective throughput of content for warfighters at the tactical edge, which will significantly enhance the tactical effectiveness of the warfighter by providing the information he needs when he needs it. In addition, information should be provided only to those warfighters with authorized access, and the information should be delivered so that the receiver can verify the integrity and originator of the content (non-repudiation).

Tactical MANETs are subject to dynamic network topologies, network partitions, energy constraints, and bandwidth limits. Current network communication is IP based and research into future Internet architectures has been exploring alternatives to IP due to the inherent limitations of its low-level of expressiveness and abstraction. ENCODERS developed algorithms that operate at and exploit the higher-level of abstraction offered by a content-based networking architecture, and developed techniques for the storage and dissemination of content that is relevant in a given context, thereby further exploiting the richness of the available metadata and user/application interests.

1.2 SRI's Team and Program Role

The SRI International (SRI) team was selected by DARPA as a Technology Developer on the CBMEN Program for the following two technology development tasks identified in the CBMEN BAA:

- **Task 3:** Managing Distributed Content on a MANET system.
- **Task 4:** Securing Content while Maintaining Accessibility.

To meet the CBMEN technical objectives, SRI developed core technologies for content-based networking and provided these technologies (and support) to SAIC, who was selected by DARPA to be a mobile system integrator (MSI). SAIC was responsible for integrating the technology developments from various technology developers (TDs) to produce functional content-based networks.

The SRI team included the following subcontractors: SET/SAIC (Phase 1 only), Suns-Tech, and GPC (Gerla Phillips Consulting). Team leadership includes: At SRI: Dr. Mark-Oliver Stehr (Phase 1 PI), Dr. Carolyn Talcott (co-PI, Phase 2 PI), Dr. Minyoung Kim (Task Lead, Dissemination and Caching), Dr. Ashish Gehani (Task Lead, Security), and Dr, David Wilkins (Phase 2 Project Manager); at Suns-Tech: Prof. J. J. Garcia-Luna-Aceves (PI), Prof. Hamid Sadjadpour (co-PI); At GPC: Prof. Mario Gerla (PI).

In Phase 1, the SAIC MSI team (led by Dr. William Merrill and George Weston) provided the development platform and successfully demonstrating an integrated CBMEN system based on the SRI ENCODERS software at Ft. AP Hill, VA, in May 2013. MIT Lincoln Laboratory performed an independent evaluation in testbed experiments and in the field.

In Phase 2, SRI's funding was cut after the first increment, so Phase 2 work was not completed. Specifically, the planned demonstrations were cut. The SAIC MSI team was integrated work for another performer, and SRI cooperated with that effort, providing software, documentation, and support as requested by SAIC.

ENCODERS is SRI's CBMEN content-based networking solution. It builds on and extends both SRI's work from the DARPA Disruption-Tolerant Networking (DTN) Program, and Haggle (http://www.haggleproject.org/) an existing modular, open-source, content-based mobile networking framework. The SRI team designed and built a

software system that incorporates technology development in the following areas, each of which will be described in its own subsection in Section 4.

1. Content-Based and Utility-Based Dissemination
2. Content-based Protocols
3. Content-based and Utility-based Caching
4. Network Coding and Fragmentation
5. Security

In Phase 1, SET/SAIC also developed technology for *Adaptive Interest Modeling* (AIM). This task was cut in Phase 2, and the details of this work can be found in our Phase 1 Final Report. Briefly, in Phase 1, ENCODERS incorporated an interest model capturing the needs of local applications that is exploited to improve network latency and information availability at the cost of using increased network bandwidth. AIM works by observing node queries, subscriptions, and publications and developing a model of the information needs of the warfighter. The interest model is used to anticipate information needs and ENCODERS can then pre-fetch relevant data objects [18].

Early on, DARPA requested that SRI develop a Value Proposition for ENCODERS and the measures needed to demonstrate the achievement of that Value Proposition. Based on the CBMEN Program Objectives, we proposed the following Value Proposition, and this proposition has informed the tradeoffs made in our design.

In the context of a tactical-edge MANET (which includes: network disruption & reconnection; and limitations on bandwidth, range, transmission power, computing power, memory), we expect ENCODERS as proposed to:

- Deliver the most relevant information available to the warfighter in a timely manner.
- Limit or eliminate the delivery of non-relevant information.
- Ensure that information is provided only to those personnel with authorized access, but without the inconvenience or overhead of key management.
- Provide an additional layer of more fine-grained security and trust by operating on top of existing, possibly certified, security solutions.

The system performance measures used to evaluate this value proposition are described in Section 5.

2 xOverview of ENCODERS Design

2.1 Overall Architecture

ENCODERS is SRI's CBMEN content-based networking solution that provides network services and transport architectures required for efficient, transparent distribution of content in mobile ad hoc networks. These services are application-independent and network-agnostic with the goal of reducing latency and increasing the effective throughput of content for warfighters at the tactical edge. ENCODERS is a search-based data-dissemination framework designed for mobile, opportunistic communication environments. ENCODERS is open sourced; visit http://encoders.csl.sri.com/.

SRI started with the Haggle open-source code base, which provides underlying functionality for neighbor discovery and basic protocols, among other things. We made major improvements in it, including improving performance in mobile networks and extensions for utility-based dissemination and cache management, semantic annotations, and security.

A brief overview of the CBMEN Program's network context is provided in this section to facilitate an understanding of SRI's Design. This overall context is depicted in Figure 1 as a network stack. The middle layers in white depict the enhancements to Haggle that are required to realize CBMEN functionality. These ENCODERS components are layer-less, a feature of our design, as described below, so should be visualized as being in a circle that operates between the levels below and above.

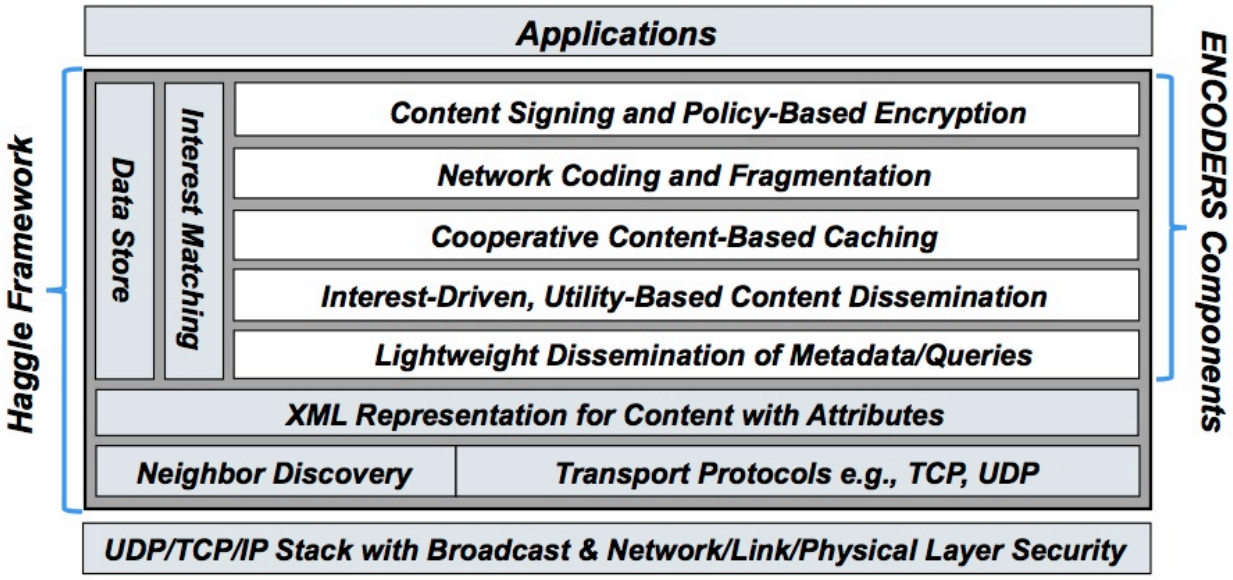


Figure 1. ENCODERS capabilities within the broader CBMEN Program context, with lower layers of the network stack below, and applications above. The white boxes are new capabilities in ENCODERS, which are layer-less.

Applications communicate with ENCODERS via data objects that contain both metadata and content. Metadata includes both content description and description of application interest. This separation of metadata from content allows the selective distribution of content based on interest matching and is a key feature supporting the efficient use of bandwidth and low latency in ENCODERS (because content is much larger in size than the metadata that describes it).

Below the Haggle framework is the network stack. This includes a traditional UDP/TCP/IP network layer as well as MAC and physical layers. ENCODERS uses UDP to connect to local applications and UDP/TCP to connect to peers. Section 2.2 gives a brief overview of the ENCODERS architecture and its leveraging of Haggle, and Section 4 describes the enhancements to Haggle that were required to achieve the CBMEN value proposition.

2.2 ENCODERS Architecture Leveraging Haggle

A key of SRI's approach is leveraging Version 0.4[1] of Haggle as the starting framework. Thus, an understanding of Haggle is essential to understanding the ENCODERS design. Haggle is an open-source framework that was implemented and funded by the European Commission under the Information Society Technologies Program of the 6th Framework between 2006 and 2010. Some of our description was abstracted from [6] and the Haggle website http://www.haggleproject.org/.

Improving performance was a main focus of SRI's extensions to Haggle and we achieved our Phase 1 performance goals. Haggle, and thus ENCODERS, is multi-platform and runs on Linux, which sped up our development cycle, especially when using the CORE network emulation environment (in Linux) for testing.

Like Haggle, ENCODERS is a search-based data-dissemination framework designed for mobile, opportunistic communication environments. This search-based approach is used for resolution (mapping data to interested receivers) and prioritization of sending and receiving data during encounters between nodes. Haggle provides underlying functionality for neighbor discovery, basic protocols, data object representation and storage, and basic interest resolution, thus removing the need to implement such features in applications.

The ENCODERS architecture, depicted in Figure 2, is event-driven, modular, and layer-less, which provides flexibility and scalability. Central in the architecture is the kernel. It implements an event queue, over which managers that implement the functional logic communicate. *Managers* are responsible for specific tasks such as managing communication interfaces, encapsulating a set of protocols, and forwarding content. The managers are represented by the wedges, labeled Forwarding, Protocol, and so forth. The managers interact only by producing and consuming events. This makes it easy to add, replace, or remove managers, as they do not directly interact with each other.

[1] More precisely, we are using the slightly more recent development snapshot that was available at program inception, but the changes are minimal.

The circular structure of the architecture in Figure 2 illustrates its layer-less design, depicting that there is no fixed ordering for execution of the managers. The kernel contains, apart from the event queue, a number of shared data structures, such as active neighbors, listening sockets, and also a data store.

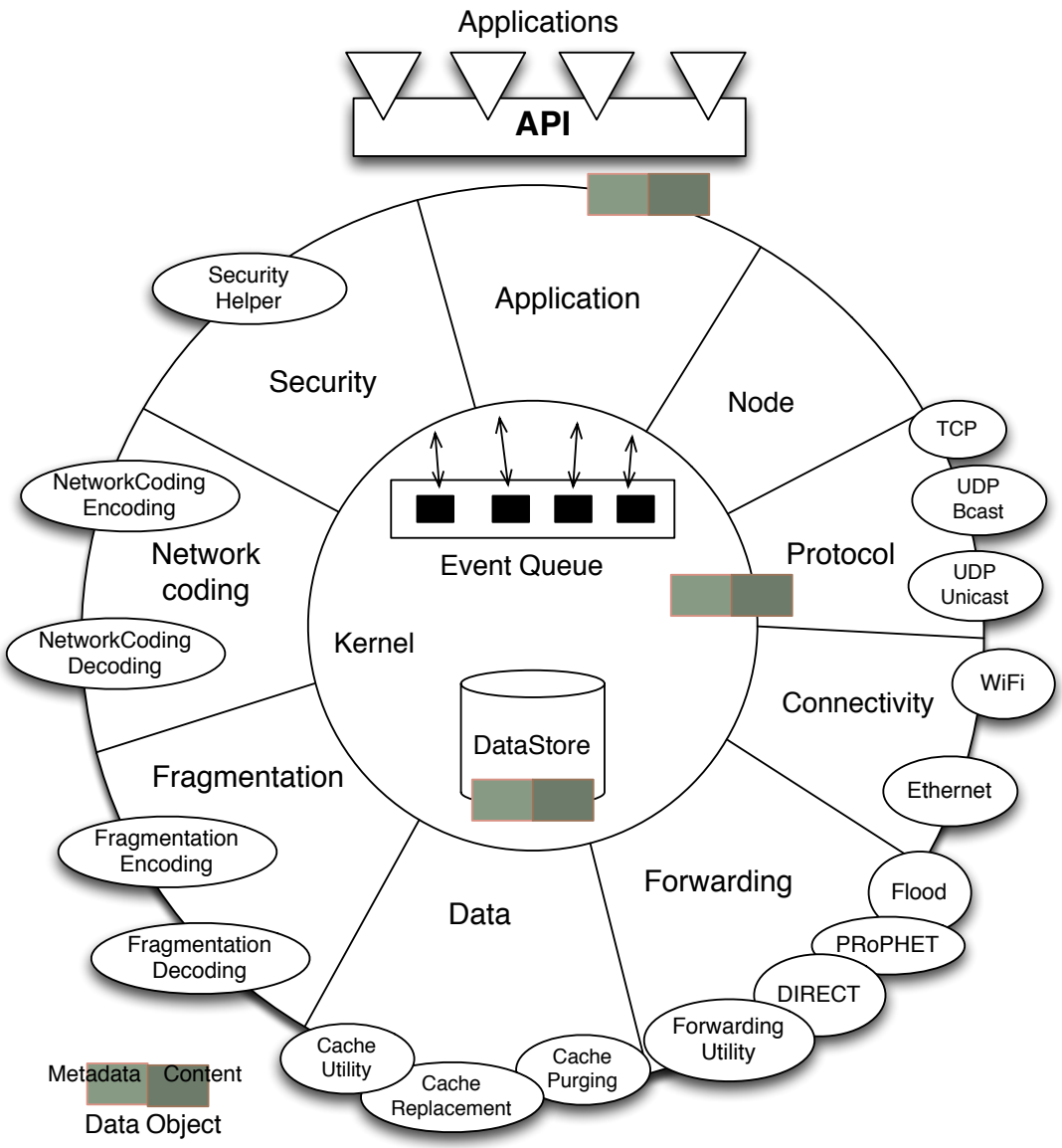


Figure 2. ENCODERS Architecture and API. The circle depicts the layer-less interaction. Each wedge of the circle is a *Manager*, and they interact through the Event Queue and DataStore in the Kernel (center). The managers may use the options provided by the modules in the small circles. Not all managers are shown.

Managers use *modules* (depicted in the figure as small outermost circles attached to certain managers) to implement specific algorithms (instead of mandating a specific algorithm for each component). Thus, the Forwarding Manager has different forwarding algorithms available in modules.

SRI implemented the new CBMEN capabilities largely as new or modified managers or new modules in the Haggle architecture. In addition, modifications were made throughout Haggle to improve performance. These are the most important extensions:

1. Lightweight dissemination of node descriptions and other small data objects such as metadata was implemented as a new lightweight flooding module in the Forwarding Manager.
2. Interest-driven content distribution was implemented as a new routing module in the Forwarding Manager.
3. Cooperative content-based caching was implemented as a module in the Data Manager.
4. A general framework for utility-based dissemination was implemented in Phase 2 as a set of modules on two new managers, the SendPriority and Replication Managers.
5. Network coding was implemented as a new standalone Network-coding Manager.
6. Fragmentation was implemented as a new standalone Fragmentation Manager.
7. Content signing and policy-based encryption were implemented as an extension of Haggle's Security Manager.

2.2.1 Functional Details

Although there are several papers and technical reports describing Haggle [6,7,8], this document describes how Version 0.4 works in practice. There are some subtleties (especially with respect to node descriptions) that we emphasize here so that we can accurately describe new ENCODERS capabilities. Most of the details described in this section are present in Haggle and were inherited by ENCODERS.

Applications express *interest* in content using a set of (key, value) attributes.
The application assigns a weight to each interest attribute that signifies the importance of the attribute. Each node supports multiple network interfaces. One key advantage is that applications in ENCODERS do not need to be aware of the underlying network technology. For example, a node may concurrently communicate with one neighbor over Ethernet and another neighbor over Bluetooth, without an application writing custom code for each technology. Similarly, one could add a new interface module (say, for RS232) to the kernel and existing applications will utilize it transparently.

A fundamental primitive is the *node description*. Node descriptions describe the state of a node at a particular instant in time. They include information such as: a list of all of the node's interests, a summary of all of the data objects currently stored at the node (a Bloom filter), a list of all of the node's network interfaces and their respective addresses, and a creation time stamp. Node descriptions are propagated throughout the network in human-readable XML format. By default, the Bloom filter is 2000 bytes and the entire node description requires 3 to 4 1500 byte packets, depending on whether the security module is enabled.

Periodically, each Haggle node issues a UDP broadcast Hello packet on all of its Ethernet interfaces to discover 1-hop neighbors. Upon two neighbors discovering each other, they will exchange their node descriptions. Logically, ENCODERS uses a single

graph data structure (the data store) to perform the matching of content and interests [10]. Data objects and their associated attributes (metadata) are stored in this data structure. Node descriptions are treated as data objects whose attributes represent the node's interests.

Data objects can be inserted in the data store either by a local application, or by receiving a data object from a neighbor. Upon receiving a new data object, ENCODERS adds it to the data store and triggers a resolution operation to build a list of all node descriptions in the data store that have an interest in common to the attribute specified by the data object. The Forwarding Manager iterates across this list of node descriptions to forward the data object to the interested nodes. If a node description is that of the local machine, then the data is passed to the local application. If a node description is that of a 1-hop neighbor, then the data is sent to the neighbor using the protocol appropriate for that neighbor's interface. Otherwise, the node description is that of an interested node which is multiple hops away, which triggers the delegate forwarding procedure, which finds one or more 1-hop neighbors to forward the data to, as a relay to the interested node.

Whenever a new node description is inserted into the data store, ENCODERS performs an analogous operation where it examines the data store for all data objects that have an interest in common to the interest specified in the node description. Using the attribute weights and resolution parameters in the new node description, a ranked list of data objects is created to send to the node represented by the node descriptions. As before, if the new node description is that of a 1-hop neighbor, then the matched data objects are forwarded directly using the corresponding protocol specified by the neighbor's interface. Otherwise, the delegate forwarding procedure finds and forwards to one or more 1-hop neighbors.

2.2.2 Subtleties

Haggle and ENCODERS are intended for the communication of larger (i.e., semantically meaningful) chunks of information, e.g., files rather than packets. They aim at a longer time-scale than traditional packet-switching networks, meaning that typical delivery times are on the order of seconds and minutes rather than milliseconds. One benefit of moving to semantically meaningful units of information is the new level of expressiveness offered by content-based networking. Hence, the selection of the right applications is important to exercise and demonstrate the benefits.

A key subtlety is that node descriptions are treated as data objects, where each node description's set of interests is treated as attributes. This functionality is not fully explained in the Haggle documentation, but SRI verified that it occurs, based on analysis of the code and Wireshark traces. Specifically, resolution is slightly different than described above for newly inserted node descriptions: when a new node description is inserted into the data store and a list of data objects is generated to push to the new node, data objects containing node descriptions are included in this list. Similarly, this new node description is also pushed to any other nodes (whose node descriptions are in the data store), but only if there is a common interest.

3 Demonstration

The SRI International (SRI) team developed core technologies for content-based networking and provided these technologies (and support) to SAIC, who was selected by DARPA as SRI's mobile system integrator (MSI). In Phase 1, SAIC (led by Dr. William Merrill and George Weston) started with SRI's ENCODERS as the core content-based networking component and integrated some smaller components from other TDs to produce a functional content-based network, working on the devices selected for the Phase 1 demonstration. SRI's funding was cut early in Phase 2, so SRI did not participate in Phase 2 demonstrations.

The SAIC MSI team provided the development platform and successfully demonstrated an integrated CBMEN system based on the SRI ENCODERS software at Ft. AP Hill, at the end of Phase 1 (May 2013). MIT Lincoln Laboratory, leading DARPA's Evaluation Team, performed an independent evaluation in testbed experiments and in the field.

In the field, data was passed between squads during austere operations. Our scenarios demonstrated content sharing when disconnected from headquarters, and content exchange in passing. Scenarios included over-the-air tests for 30 Android phones, moving in scenarios for 3 squads. All evaluations were for stressed networks, with limited bandwidth, short connection times, and/or lossy connections.

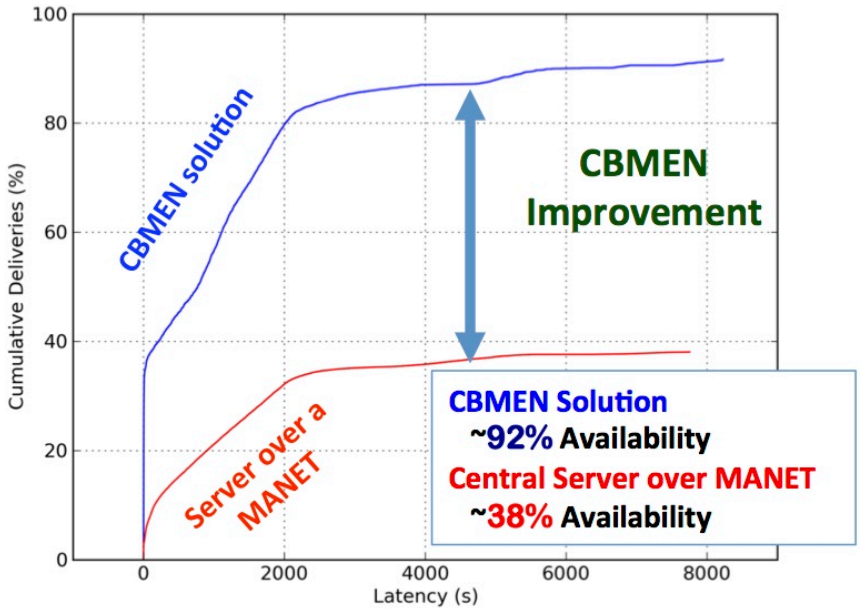


Figure 3. Content delivery, latency, and availability from Government Austere-Squad Field Test on 5/16/13. A system based on ENCODERS outperformed a Government-defined baseline Tactical Central Server. Content is pulled even when its publisher is disconnected, and is proactively pushed before subscribers ask for it. [Source: SAIC, used with permission.]

As shown in Figure 3, SRI's solution had 2X - 2.5X times higher availability than the baseline centralized system (defined with the Government Test and Evaluation Team). Our solution's average latency is one-tenth of the baseline, and the CBMEN solution used 10 to 57% fewer resources (depending on the scenario) than the baseline.

To summarize the on-the-air tests, SRI ENCODERS showed:

1. 10x reduction in latency
2. 2-2.5x increase in content delivery
3. while using 10-57% fewer resources, depending on the scenario.

SRI's extensions to Haggle include utility-based cache management. Utility-based caching is a generalized caching approach that frames the cache purging and replacement decisions as a utility-optimization problem. A caching policy defines content- and context-sensitive utility functions that vary in time and space.

Cache management helps ensure that resources are primarily used for content that is relevant to the user and that the amount of content managed by the network does not grow beyond its capacity. Cache management is also essential for store-and-forward content dissemination if no end-to-end physical path exists between a source and a destination.

Various strategies are available for content-based caching in ENCODERS, and their effects on delivery rates in austere networks is shown in Figure 4. The performance of ENCODERS cache purging options is shown. ENCODERS increases delivery rate (by 7x in some cases) and decreases latency in typical small unit operations. The 7x increase occurs when ENCODERS employs its smart replacement of the cache (e.g., we cache only the latest position in Blue Force tracking). While the concept is very general, the most common use is keeping only the freshest piece of content, while stale content is discarded from the cache (e.g., current blue-force position updates replace earlier data).

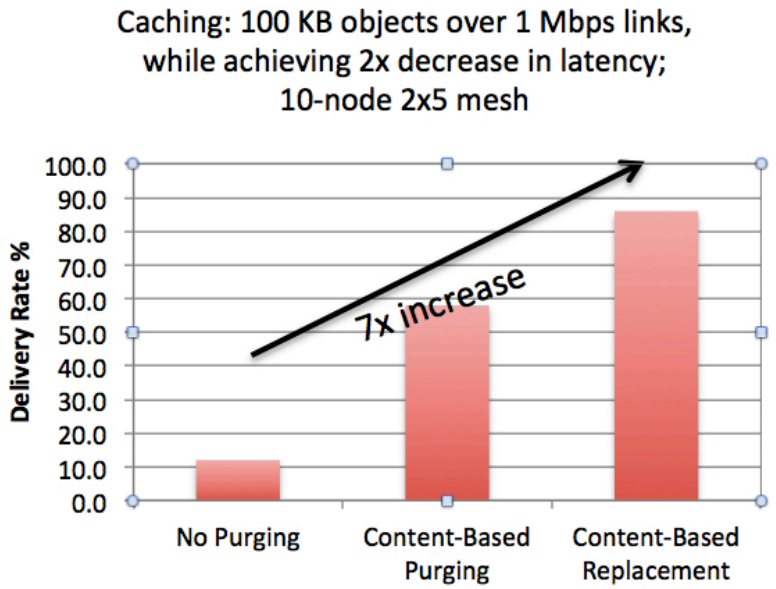


Figure 4. The performance of ENCODERS cache purging options is shown. ENCODERS increases delivery rate (by 7x in some cases) and decreases latency in typical small unit operations.

We enhanced Haggle with a Network-coding Manager that uses a technique that fragments and encodes content for dissemination in the network. Its purpose is to provide robust (and hopefully efficient) dissemination in networks with unreliable node connectivity. ENCODERS also provides a standard TCP-IP-like fragmentation option. Network coding is used on the data content for files larger than a specified minimum file size, but *not* on the attributes (so that the attributes can still be matched with interests), and not on the relatively small node descriptions. The results of evaluating our network-coding option at Fort A.P. Hill are shown in Figure 5.

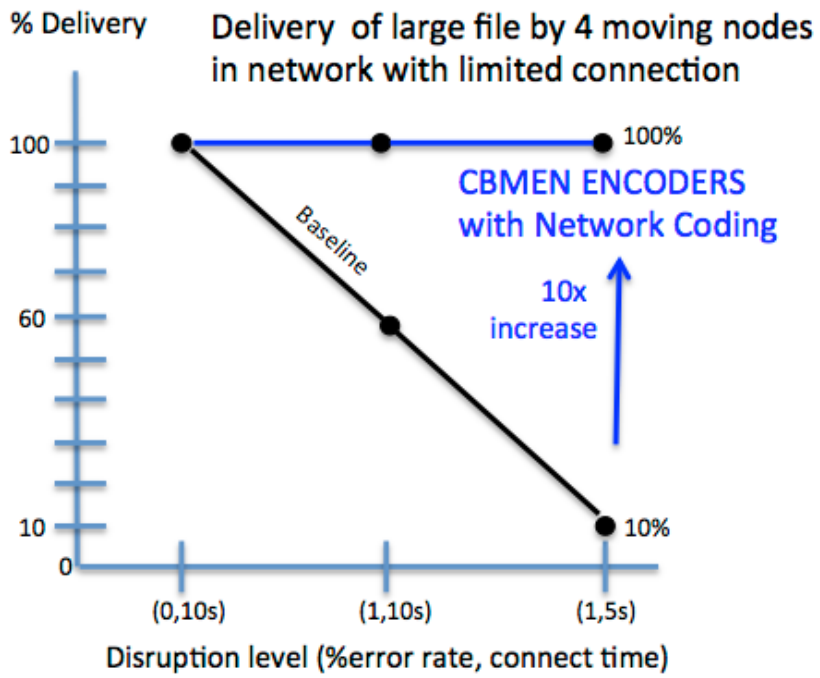


Figure 5. The performance of network coding, which leverages aggregate bandwidth of a group of nodes to deliver large content items in austere conditions.

SRI and its team members prepared four sidebar presentations at the VIP day on May 29 at Ft. AP Hill, VA. The sidebar presentations focus on the value for the warfighter, specifically how CBMEN allows them to extend their operational capabilities. Each technology sidebar had a demonstration associated with it, which used the actual ENCODERS system underneath. A lot of effort was devoted to fine-tune the presentations with the help of our subject matter experts. SRI's overview of the sidebars is shown in

Figure 6. More details can be found at the ENCODERS open-source website, http://encoders.csl.sri.com.

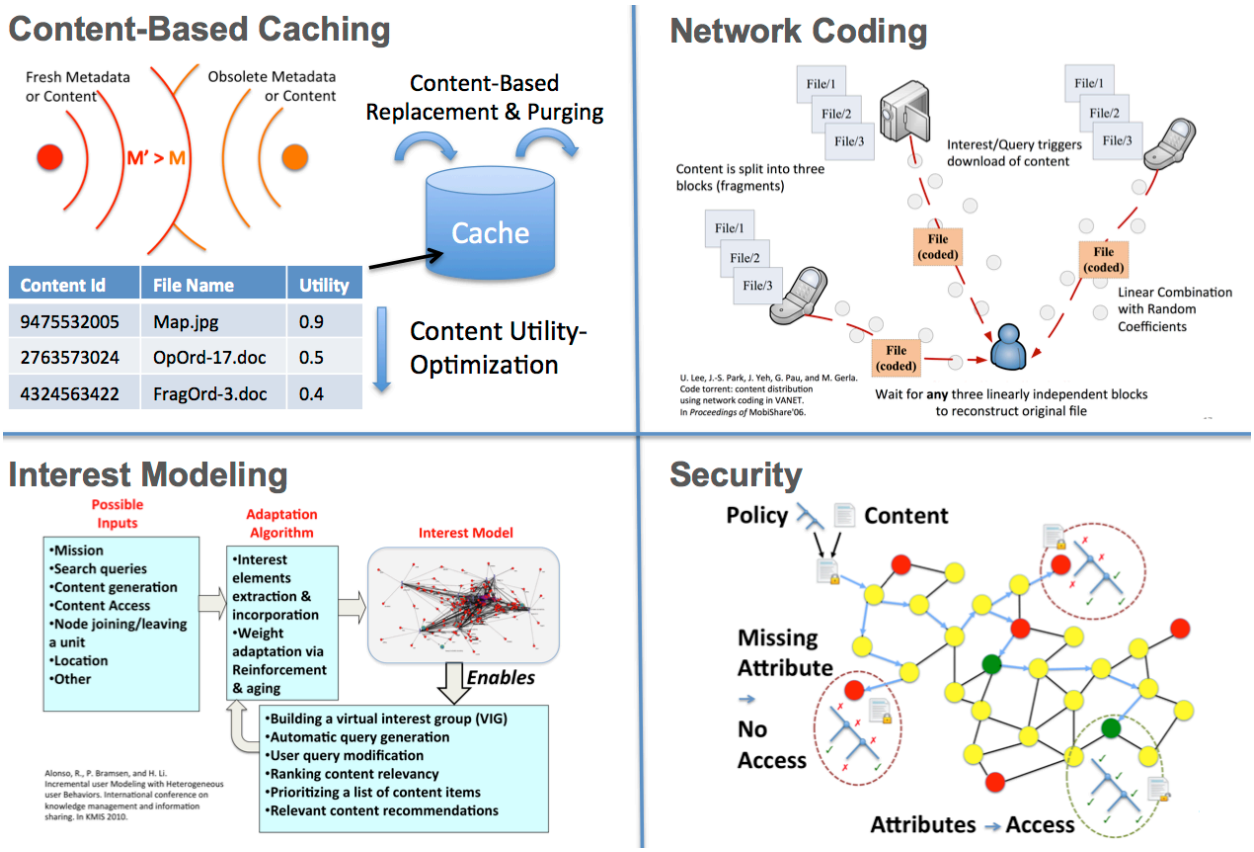


Figure 6. Our team's four Phase 1 sidebar demonstrations using ENCODERS. Content-based caching was developed by Suns-Tech, Network Coding by GPC, and Interest Modeling by SET. The Security sidebar demonstrated SRI's security solution.

4 Technical Solution and Accomplishments

The technology development performed by the SRI team can be grouped into areas that extend the Haggle framework in orthogonal dimensions. A separate subsection will cover each of these areas.

- **Content-Based and Utility-Based Dissemination.** The Haggle enhancement that provides interest-driven content distribution is based on DIRECT [10], which was developed in the DARPA DTN Program. It extends Haggle by providing a mechanism that supports lightweight dissemination of node descriptions (including node interests) and dissemination of content, which is typically more heavyweight, when new interests are detected. Our new generalized content-distribution architecture supports utility-based dissemination to prioritize the content dissemination for better use of limited network bandwidth.

- **Content-Based Protocols.** We refactored the Haggle protocol design to improve performance by implementing additional UDP-based protocol options.

- **Content-Based Caching.** The basic cache management strategy is based on opportunistic caching and orderings, ensuring that the purging and replacement strategies are not ad hoc, but inherently content-based, and intuitive concepts such as relative prioritization and utility can be expressed for efficiency and timeliness of content delivery. Our approach to caching is to treat it as a specialized utility-optimization algorithm. We extended Haggle with configurable caching strategies that provide a generalized mechanism for handling data that has been marked with specific tags by the user. Utility can also consider military social structures and be used for smart replication to enable cooperative caching in the tactical edge.

- **Network Coding and Fragmentation.** Network coding is a technique that fragments and encodes content for dissemination in the network. Its purpose is to provide robust (and hopefully efficient) dissemination in networks with unreliable node connectivity. We enhanced Haggle with a Network-coding Manager that operates on larger data content but neither on the attributes (so that the attributes can still be matched with interests) nor on the relatively small node descriptions. Use of network coding may be further specialized, e.g., its use may depend on various aspects of the situation (context-aware), including network state. ENCODERS provides the option of fragmentation, as an alternative to network coding.

- **Security** in ENCODERS provides data integrity, non-repudiation, and confidentiality. Our approach replaces the implementation of digital signatures in Haggle with a new multi-authority certification framework for signatures and adds a new layer of attribute-based encryption to cryptographically implement complex security policies framed over attributes certified by multiple authorities. Section 4.5.4 describes the concept of operations (CONOPS) for small unit operations for the ENCODERS security solution.

4.1 Content-Based and Utility-Based Dissemination

At a high-level, Haggle is an application-layer (although it supports a variety of layer 2, 3, and 4 technologies) content-centric framework for pocket switched networks [5]. Applications express interests in particular types of content, and they also add content with metadata to the network. The role of Haggle is to efficiently send content in the network to interested nodes. To match content to interests, Haggle disseminates the interests of nodes throughout the network, and then disseminates the relevant content to interested nodes.

ENCODERS provides a mechanism that enables much more flexible content dissemination than is provided by Haggle, as described in this section. Dissemination in ENCODERS involves the Node, Data, Forwarding, Interest, SendPriority, Replication, and Application Managers. These in turn rely on content-based protocols, which use the Protocol and Connectivity Managers, which are described in Section 4.2.

4.1.1 Interest-Driven Content-Distribution Approach

We first examine content dissemination and discovery. Unmodified Haggle can support the following methods of content and interest dissemination, either explicitly or implicitly by assigning interests to certain nodes: 1) epidemic dissemination (flooding) of content and interests (all node interests and content share a common attribute); 2) single-hop dissemination (no routing) of interests and content (disable delegate forwarding); and 3) multi-hop encounter-based dissemination of interests and content using Prophet delegate forwarding [5].

To improve content dissemination, we implemented a new module for the Forwarding Manager named DIRECT (DIsruption REsilient Content Transport), which is inspired by a content-dissemination protocol of the same name for disruption-tolerant networks [14]. DIRECT offers an alternative to the three Haggle methods as it disseminates interests epidemically, but unicasts content from source(s) to the destination. Presumably, interests are smaller in size than data, so this approach may utilize network resources more efficiently than an approach which floods content. By implementing DIRECT, we enabled a side-by-side comparison with Prophet (see our Phase 1 Final Report).

In DIRECT (as in Haggle), nodes publish objects and assign attribute metadata to each object. If a node wishes to receive some piece of content, it issues a query that expresses the interest in terms of attributes. DTN DIRECT supports 3 different types of queries: *pub-match*, *all-match*, and *stop-match*. In pub-match, the query will only be answered by the publisher. In all-match, every object that matches the query will be sent back. In stop-match, the first object that matches will be returned and the query is not re-broadcast. There is no guarantee that only one data object will be returned in a stop-match. ENCODERS implemented only all-match in its DIRECT module (leaving the others for future work), because this is closest to the semantics of Haggle's standing subscriptions. However, we see an opportunity to exploit the stop-match mode, e.g., to access data objects with lower overhead based on their unique identifier. Stop-match may also be of interest to optimize certain one-shot queries for which only a single result is expected or needed.

Design and Implementation. To evaluate DIRECT's effectiveness, we implemented multiple versions: Alpha-DIRECT and Alpha-DIRECT++. These are discussed in detail in the ENCODERS Software Design Description. To support a large class of scenarios and improve performance, we implemented an extensible mechanism of distributing content proactively, without explicit user subscriptions.

We developed Alpha-DIRECT++ in response to the limitations of Alpha-DIRECT (see Design Description). We added periodic node-description refresh, and node description purge. We made fewer calls to the database when disseminating node descriptions, by adding an in-memory node store cache, which contains all discovered node descriptions (not just one-hop neighbors). We extended the flooding mechanism primarily used for node-description dissemination to support a basic form of proactive replication. We also integrated fragmentation and network coding to work with DIRECT. This combination offers greater content-dissemination performance in networks with unreliable, short-lived paths.

4.1.2 A General Architecture for Content Distribution

SRI refactored and generalized Haggle's forwarding architecture. Based on our DTN work on interest-driven routing [14], we distinguish lightweight knowledge about the network and application content, and then use different dissemination mechanisms for each of these classes. We took this idea one step further and extended the Forwarding Manager to allow the user to configure a Content Classifier module and one or more forwarder modules that can register for a particular content class. As an example of content classification, our *basic classifier* assigns one of two possible tags. It assigns the "lightweight" tag for node descriptions, and the "heavyweight" tag for all other data objects. This allows ENCODERS to choose forwarding protocols dynamically based on the content to be forwarded. Currently, there must be exactly one forwarder module for each content class, or a default forwarder that is used for content not covered by the classifier. Classifiers can be arbitrarily composed, using the *priority classifier*, to form more complex rules.

DIRECT and Prophet are forwarding modules that are typically registered for heavyweight content. We added a Flood module that is typically registered for lightweight content. It forwards content to all of the node's neighbors, and relies on the Bloom filters to avoid routing loops and redundant transmissions.

Proactive Replication. We generalized ForwarderFlood to support two different types of flooding for node descriptions and arbitrary data objects: contemporaneous flooding with and without push on contact, which is a form of proactive content replication: data objects are proactively pushed to nodes that may not have any interest in the data, for the sake of increasing the delivery ratio, or reducing the delivery latency. This feature can deliver content (albeit at the expense of an increase in overhead) in scenarios where DIRECT and Prophet cannot; for example, in one-way message ferrying.

To enable the periodic dissemination of interests and Bloom filters, we extended the Node Manager to read new configuration parameters that specify a node description's refresh period and jitter. Using these, nodes periodically broadcast their own node

descriptions even if their state is unchanged, with the jitter being used to avoid synchronization problems (see the Configuration Manual). Additionally, we updated the Node Manager to purge node descriptions that are older than a maximum node description age (based on creation time), which prevents the maintaining of unnecessary state and routing on paths that are no longer valid.

To avoid frequent database accesses due to node-description propagation, we added an in-memory cache that sits directly above the database and is used solely for storing node descriptions. The node descriptions that are not within the immediate neighborhood of a particular node are also stored in this cache. This is a generalization over unmodified Haggle, which cached only neighbors in the node store.

Prioritized Content Dissemination. The purpose of the SendPriority manager is to maintain an ordering on the data objects to be sent. This ordering is formulated as a partial order, specified in the configuration file. Partial orders can be composed of other partial orders, and different types of partial orders can be combined to define flexible prioritization policies. Through these policies, the network operator can ensure that high-priority content is delivered before low priority content. This mechanism is important when links have limited bandwidth and are connected for small durations—the most important data should be sent first. This manager intercepts data objects before they are placed on the protocol queue, and it maintains its own queues that are drained into the protocol queue.

Utility-based Dissemination. The purpose of the utility-based dissemination is to provide a system that supports intelligent replication policies and is flexible. Similar to how the utility-caching system frames the caching problem as a 0-1 knapsack optimization problem, this system frames the replication problem as a 0-1 knapsack optimization problem. It replicates content based on the utility of a certain content to a certain node.

The utility-based dissemination pipeline executes the following steps when any neighbor node changes, any new data objects are inserted, or a specified poll period transpires:

- A utility is computed based on both the data object and potential destination node. This differs from the caching pipeline, which computes utilities solely based on the local node.
- A threshold is set, and any data objects with utilities above this value is subject to replication.
- A knapsack optimizer will consider data objects above a threshold for replication to a node, and the estimated connection time to that node. The optimizer considers which data objects have the highest marginal benefit (i.e., utility/size), while still fitting in the knapsack and being bound by the estimated contact time.
- It has an upper bound (the maximum number of tokens) of how many data objects can be queued for replication at a time. Upon every success or failure of replication, if a node is still a neighbor, a token is put back, and the utility pipeline is called for the same destination node. Optionally, instead of keeping tokens per

node, the tokens can be measured against total number of data objects to be replicated.

Replication is considered low priority, thus, it should not interfere with higher priority attribute matches.

Semantics and First Class Applications. In ENCODERS, we modified Haggle to consistently use its mathematical semantics at the application level, which is what matters for end users. In ENCODERS, application nodes become first-class citizens (as opposed to be approximately represented by their proxy, the device node). Aggregation is disabled in favor of accuracy. This feature is called *first-class applications***.** See the ENCODERS Software Design Description for details of the semantics and the Configuration Manual for how to enable it and how to test queries.

4.1.3 Limitations and Future Work

Content dissemination in ENCODERS addresses several limitations of previous versions, but not all of them. There are some inherent disadvantages related to the underlying transport protocols. These are discussed in the ENCODERS Software Design Description, along with suggested directions for future work. We believe that further experimentation is necessary to evaluate the effectiveness of Bloom filters and the trade-offs of their use with respect to node description propagation and interest specificity.

4.2 Content-based Protocols: Lightweight Unicast and Broadcast

SRI refactored the Haggle protocol design to improve performance in the unreliable broadcast scenarios that CBMEN targets. Specifically, we implemented and used additional UDP-based protocol options.

Unmodified Haggle supports several protocols for transporting data objects between a sender and receiver pair. The decision of which transport protocol to use is made after the Data Manager and the Forwarding Manager have determined what content to send, and where to send it. Unmodified Haggle has a simple policy that chooses the transport protocol based on the interface type. For example, unmodified Haggle will only use TCP communication between hosts that are connected through an Ethernet interface. It employs a simple *stop-and-wait* protocol on top of the underlying network transport, to ensure data delivery and to reduce redundant transmissions. (Both the stop-and-wait protocol and the network-layer transport protocol only occur between a sender and receiver pair when they are neighbors.)

While this design may be effective in pocket-switched networks where contact times are long and infrequent, it must be adapted to efficiently support unreliable broadcast environments. For example, DIRECT uses broadcast messages to propagate interests, and will be inefficient when TCP is used as a substrate (especially if the neighborhood is large). Additionally, protocol selection should be based on the current state of the environment, and the content to be sent. To this aim, we modified the protocol

architecture to include unreliable UDP-based protocols, and we added a classification system that uses the content to dynamically select the transport protocol.

The stop-and-wait protocol works as follows: the sender initiates the communication by first sending the data object header. The receiver then uses the header to send either an ACCEPT or a REJECT to the sender, depending on whether the data object already exists at the receiver. If the receiver sends a REJECT, then the sender assumes that the receiver already has the content, and marks the content as delivered. Otherwise, the sender proceeds to send the data object. Upon successfully sending the body of the data object, the receiver will send an ACK message to acknowledge receipt to the sender. In the case of TCP network transport, these higher-level control messages operate in conjunction with the TCP control messages.

After the sender receives the ACK, it may send subsequent data objects using the same method. In the case of an error, there are protocol parameters that allow Haggle to retry this same process to send the data object (without keeping track of what has already sent) after a specified delay. From an implementation perspective, communication only occurs between a sender and receiver instance, thus if two nodes are exchanging their node descriptions there will be two separate TCP sessions. We parameterized many timers and retry counts used by this protocol, enabling users to specify them in the configuration file.

4.2.1 Lightweight Unicast and Broadcast via UDP

As a step towards UDP broadcast, we implemented a UDP unicast protocol that is an alternative to TCP. Unmodified Haggle has a UDP protocol, however it is insufficient for our purposes, because it is designed to be used for communication only between local Haggle applications and the Haggle daemon. Also, unmodified Haggle uses UDP for its neighbor discovery protocol, however it is not general enough or compatible with the existing protocol architecture.

Our new protocol leverages the existing protocol architecture, and uses the same control protocol, but it does not have the network-protocol overhead that TCP incurs due to its congestion control and reliability requirements. In ENCODERS, when the protocol sends the header of the data object (or body), we pass the entire header (or body) to the UDP socket for transport. Thus, we rely upon the kernel's use of UDP fragmentation for delivery. This approach of relying upon implicit fragmentation may be ineffective in lossy channels, because an error, loss, or out-of-order delivery of any individual fragment prevents delivery of the entire header (or body). However, these limitations are less pronounced due to the hop-to-hop nature of the communication.

Our implementation of UDP unicast can be a useful alternative to TCP for small data objects and can be used in parallel with TCP with our new protocol architecture. However, the real advantage of UDP is its capability to naturally generalize to broadcasting, which is a natural fit for content-based wireless networking.

We initially implemented UDP broadcast for node descriptions. This feature is critical for data dissemination that requires flooding, such as DIRECT's use of flooding interests.

Our implementation bypasses the higher-level stop-and-wait protocol, which is possible because unmodified Haggle treats an entire node description as a data object header. After broadcasting a node description, we optimistically assume that the data object was successfully delivered to all known neighbors. To simplify the design, the protocol has a particular receiver in mind (the "reason" for the broadcast), but this receiver is not indicated in the data that is actually transmitted over the wire.

4.2.2 Limitations and Possible Future Directions

The biggest limitation to our approach is the inherent unreliability of UDP. We may explore alternatives to the stop-and-wait protocol that provide reliability and can exploit broadcast to gain additional performance. Our UDP-broadcast implementation should be extended to support broadcast of arbitrary data objects, and explicit fragmentation, which will increase network utilization by not discarding overheard information. For UDP unicast, our approach of passing the entire data object body to the UDP socket for implicit fragmentation limits the size of the data objects that can be sent. Thus, UDP is practical only for small data objects. The rapid increase of packet losses due to UDP fragmentation is a well-known limitation of both UDP unicast and broadcast. The use of TCP avoids fragmentation by only sending packets smaller than the MTU and hence will be advantageous in many cases up to a certain number of neighbors.

Additional experimentation is required to evaluate the ramifications of broadcast interacting with the link-layer on phones. In particular, broadcast may decrease battery utilization depending on its interaction with link-layer scheduling. It will certainly lead to an overall increase in the amount of packet processing by each phone. When implementing the proposed design, several unexpected problems arose with UDP and link-layer interactions, which led to poor performance. The ENCODERS Software Design Description has details of our solutions to these problems.

Another limitation of UDP broadcast in 802.11 wireless networks is the lack of RTS/CTS signaling and link-layer acknowledgements, which makes it inherently less reliable than UDP unicast. Interestingly, the corresponding losses increase with medium contention and hence with the number of neighbors. ENCODER's flexible protocol architecture allows us to avoid an a priori commitment to a particular protocol and allows a future design where the protocol choice can be made at runtime, based on a range of factors and optimization objectives. UDP broadcast could also be improved by implementing a hybrid between our UDP broadcast and unicast protocols. Another promising direction is inserting an additional layer above UDP, like NORM, which provides better reliability though forward error correction and optional (negative) acknowledgements.

Currently the decision of which protocol to use is only determined by examining the data object to be sent. A useful extension would be to expand our classifiers to take into account the environment, history and resources available between the communicating nodes. The classifier architecture parallels that of the forwarder classifier architecture, and we plan to eventually merge these into a single set of classifiers.

4.3 Content-Based and Utility-Based Caching

Because content-based networks can persistently store large amounts of content, a key concern is developing mechanisms to ensure that (a) the amount of content managed by the network does not grow beyond its bounded capacity, and (b) resources are primarily used for high-utility content that is relevant to the user. Unlike IP-based networks, content-based networks maintain explicit information about the meaning and use of content at a level of abstraction corresponding to semantically meaningful units that can be exploited for better content-management.

We designed efficient and effective content-aware caching algorithms inside the Haggle framework. Our algorithms are opportunistic (or reactive), meaning that they will make caching-related decisions for content that is already located or flowing through the node on which they are running. The same algorithms can be applied in conjunction with proactive dissemination of content. When more content is flowing through the network, it is important to make fine-grained decisions about what content to cache and to quickly discard content with insufficient utility. This decision is based on the benefits and costs for each individual piece of content in a context-dependent and cooperative manner.

4.3.1 Content-Based Caching Approach

Unmodified Haggle provides a mechanism to specify priority of an interest using "weights". This mechanism is useful for prioritizing which data objects are sent first from a sender to a receiver, or limiting the number of data objects that are sent. Beyond the weights, Haggle does not natively support a mechanism for a user to specify the relevance of a class of content, or a way to specify the relative "freshness" of two pieces of content in the same class.

Caching the right information is key to improving latency when there is limited bandwidth and potentially costly content-query operations. Thus, a more fine-grained mechanism is needed to give users control over caching. The benefits include reducing network load (irrelevant data will be discarded sooner), device storage usage, and latency (more bandwidth will be available, and fewer data objects will be in the data store). Caching techniques and their trade-offs have been studied extensively in research (such as [13]). ENCODERS supports different caching algorithms from this body of research, and its design fits naturally within the Haggle framework.

In the same way that the Forwarding Manager allows a configuration to specify a forwarding module, we added a Cache Strategy module[2] to the Data Manager, which was augmented to pass a newly received and verified data object to Cache Strategy.

As with the forwarding modules, this module is asynchronous, and runs in its own thread so that the additional processing does not block the Data Manager or any other managers. The Cache Strategy module has its own event queue to serialize database accesses, which may not otherwise be atomic. For example, when a new data object is received which subsumes an existing data object, the module must delete the stale data

[2] The Technical Documentation gives the diagram and describes the classes.

object and insert the new one in a single atomic operation to avoid consistency errors such as stale data objects, or duplicates of the same data object.

To support the Total-order Replacement module, we augmented the data store and SQL data store to provide a helper function that returns all of the data objects for which the replacement module is responsible, sorted by freshness. The Total-order Replacement module is responsible for deciding whether or not to insert a newly received data object, and what data objects to remove. Additional query and query result classes have been added to the data store to allow the replacement modules to communicate with the data store and the Data Manager across callbacks.

Independent of replacement, we added cache-purging modules to provide even finer control of caching, based on expiration time. Data objects are evicted from the data store after they have surpassed a specified age. Unmodified Haggle does not purge content, except after reaching a global maximum age. Our design requires the purging module to support timer-based events for already cached data objects (not just data objects being added, as in replacement).

Content-Based Caching Strategy. To solve these Haggle limitations, we added a generic module to the Data Manager, the Cache Strategy module, which currently has *cache replacement* and *cache purging* sub-modules (shown in Figure 2). Cache Strategy is responsible for handling data that has been marked with specific tags by the user, through use of attributes. Our design is extensible so that one could easily add a module that treats Blue-Force tracking data differently than FTP or web content. Specifically, Blue-Force tracking data may have requirements such as low latency, low jitter, and faster aging (retransmissions are not necessarily useful).

The cache replacement and purging sub-modules specify the tag that identifies the class of content that should be handled by the module (see the Configuration Manual). Upon receiving a new data object, the Data Manager will determine if the cache strategy is responsible for the data object, which in turn will query the cache replacement and the cache purging components.

Content-Based Replacement. If replacement is selected for handling the content, Cache Strategy passes the newly received data object to it. The replacement module decides whether to insert or discard the data object. If the data object is inserted, then the module is responsible for deleting existing stale content from the data store.

While our architecture is open-ended to support multiple types of replacement policies, our first implementation uses a total-order replacement algorithm. This keeps only the “freshest” piece of content, while “staler” content is either dropped or deleted from the data store. The configuration file specifies a tag name (which denotes that a tagged data object is totally ordered), an id attribute (which indicates the content originator), and a metric field name (used to determine the freshness of the object).

We implemented a Priority Replacement module, which can be composed of multiple total-order replacement modules. This functionality allows customizing the configuration file to specify complex replacement rules, such as lexicographical ordering (see the Configuration Manual for an example). More general orderings are conceivable and of interest, including the most general case of a user-defined partial order as in our partially ordered knowledge-sharing model [15].

Content-Based Purging. Content can be purged either by absolute or relative expiration times. An absolute expiration time is specified as a predefined time (e.g., Jan 30, 2013 at 5:15pm). A relative expiration time is specified as a minimum time to live (in seconds) from the reception, which allows data objects reasonably sufficient time to propagate yet aims to control the amount of cached content, e.g., to reduce access times for incoming data objects. Purging enables Haggle to delete data (when no longer needed) and to provide better use of resources. Examples of this are stale orders (e.g., inactive missions) and obsolete information (e.g., old map, location, communication codes).

Content-Based Priority Policies. The ENCODERS cache-replacement modules can be prioritized when composed into a complex cache strategy. When a data object is examined to determine which replacement module to apply, priority determines how to proceed. The highest-priority replacement module that is applicable for both the data object and any specified attributes is responsible for the data object. For example, we have replacement mechanisms with priority policies based on mission timestamp and content-creation time.

Content-Based Parallel Policies. The cache-replacement modules operate sequentially to make caching decisions (because they may depend on each other), while the cache-purging modules operate in parallel (they are considered to be independent of each other). Thus, the data object can be acted upon by each of the policies, and is purged when one (or more) of the parallel policies dictates a purge. Purging can be triggered for many reasons such as creation time, expiration time, content type, or locality of content (e.g., out of scope relative to a mission or a named geographic area). Expiration time and locality are independent measures, so they can be checked in parallel for the same data object. The Configuration Manual illustrates a purging mechanism with parallel policies using absolute and relative expiration times.

Content-Based Lexicographical Replacement. We implemented lexicographical rules by using the prioritized replacement policies. When a data object is received, it uses the priority to decide which algorithm may act upon the data object. If multiple cache-replacement orderings are applicable to a data object, the priority policies can be used to define a lexicographical ordering. The replacement order with the highest priority first examines the first element to be ordered. Subsequently, replacement ordering with the n-th highest priority examines the n-th element. The Technical Documentation gives a detailed example. To avoid ambiguity, it also gives a mathematical formulation for the composition of total order replacement and priority replacement.

4.3.2 Utility-Based Caching Framework

The goal of utility-based caching is to find the optimal set of data objects that fit in the fixed-sized data store. The idea is to periodically compute a utility (a real between 0 and 1, where 1 has highest utility) and evict data objects from the cache that do not meet a sufficient utility threshold, or evict data objects with the least utility in the case where the capacity is exceeded. This mechanism allows ENCODERS to more intelligently manage disk space resources.

The strength of our method lies in combining multiple sources of knowledge to influence caching decisions, and the ease and flexibility in specifying the utility function (declarative approach). Previous strategies (e.g., total-order replacement and time-based purging) now become special cases of utility-based caching.

The caching algorithm (executed either periodically, or in response to an event that receives a new data object) is as follows:

- Compute a utility for each data object in the cache using a specified utility function.
- All data objects with utility less than a specified threshold are immediately evicted, regardless of the cache capacity.
- If the watermark cache capacity (a soft constraint) is exceeded, we frame the problem of selecting which data objects to keep as a 0-1 knapsack problem, with the utility as the benefit, and the size of the data object as the cost. A knapsack optimizer calculates the data objects to evict.

We support the described caching pipeline with several utility functions, which are listed in the Technical Documentation. We implemented a fixed-weight global optimizer, and a heuristic-based knapsack optimizer, which greedily fills the knapsack using highest marginal utility (utility/cost) first (until the watermark capacity is surpassed).

Our design of utility-based caching is used with the utility-based dissemination approach to naturally generalize to proactive and cooperative caching. Proactive caching is implemented via utility-based dissemination of content, via suitable utility functions, which take into account the benefit and cost of replication. Cooperative caching is implemented as a special case of utility-based caching and can furthermore take into account the spatial distribution of content that is already cached elsewhere, e.g., by inspecting the Bloom filters of other nodes within social groups (see Technical Documentation for social-aware utility functions).

4.3.3 Applications of Content-Based Caching

Total-order-based replacement is useful for applications such as GPS location tracking, situation-awareness (e.g., latest data from a stream of time-stamped images of a location), and real-time measurements (e.g., latest temperature of a device). For example, suppose a Blue-Force tracking application on each phone periodically sends out a location beacon containing GPS information to every other phone in the network. The application only cares about the most recent location, and it uses total-order replacement to specify this preference to ENCODERS. The total-order replacement module will then discard beacons that are not the most recent for a node. Similarly, upon receiving a more recent beacon for a specific phone than what was previously

received, the module will replace the beacon data object. In this way, old beacons are removed at the earliest possible time from the network (rather than by applications at the end points). Thus, they consume fewer resources, while new beacons are given priority and have more resources to propagate.

In this context, expiration times can be used for purging sensitive content. When a soldier loses his/her smart device that contains traces of Blue-Force locations and movements, purging based on expiration times can prevent information from being revealed, even if security was compromised. Enforcing content purging based on an expiration time that is specific to the type of content can help to limit the leakage of sensitive content independent of other cryptographic security mechanisms.

Furthermore, by combining replacement with purging, a mechanism similar to a watchdog timer can be enforced. Pre-existing orders with an expiration time can be extended, in case the mission is active longer than expected. Protecting communication codes (which in many cases are associated with expiration times) can be another use-case. In a situation when a communication code is broken, one needs to resend a newer code that will replace the old code regardless of its expiration time.

Lexicographical ordering in our cache-replacement framework can be used to implement fragmentary orders. For example, the update priority can be the first attribute to be compared and the update time can be the second attribute in the lexicographical ordering. If the commander publishes content for a fragmentary order with <priority, time> attributes, then the higher priority updates will replace updates associated with lower priorities. For updates with the same priority, the update time determines which content is replaced (i.e., more recent updates replace older ones, within the class of orders that have the same priority). Other content that may benefit from this technique include security call signs and real-time intelligence updates (e.g., a mine field location).

4.3.4 Data Store Optimization and Memory Usage Control

We optimized ENCODERS' Data Store for more efficient data querying than Haggle's vanilla SQLDataStore. It contains the in-memory database that does not use SQL and is optimized for queries with a few interests (< 5) and data objects with a moderate number of attributes (< 10). This implementation is designed to support a large number of data objects (~20,000) where the number of data objects that match a particular interest is small (<10). The new implementation supports the same DataStore interface as SQLDataStore. We examined all of the queries that the DataStore performs and added multiple hash tables that act as indices for fast lookups. Data objects can be saved to and loaded from disk by using the existing SQLDataStore.

In addition, we implemented a mechanism to manage the memory usage of ENCODERS' Data Store. Because our target devices (e.g., mobile phones) usually run with an in-memory database for performance reasons, it leads to a heavy burden on the memory of the device. ENCODERS keeps attributes of data objects for faster response, adding to the memory burden. To manage the memory usage of in-memory Data Store, we created a memory constraint, which works in the following manner:

- Create a watermark for 'x' data objects, maximum, in the in-memory database.
- If in-memory database exceeds 'x' data objects, it evicts data objects below the watermark based on utility.

4.3.5 Limitations and Possible Future Directions

The ENCODERS Total-order Replacement module only keeps the “freshest” data object for a particular class of content. It may be beneficial to generalize this module to support a window of content, such as the *n* freshest data objects. Such functionality may be useful to calculate estimates such as the speed and direction of a tracked device from GPS data. Other replacement modules might include replacement by hop-count or GPS location, and the ability to specify a maximum number of data object replicas in the network. In utility-based caching, an interesting special case is content that has missed an optional (soft) delivery deadline (where the deadline can be content dependent) and does not have much utility remaining. Such content could be purged to free resources.

4.4 Network Coding and Fragmentation

Fragmenting files is a well-known and indispensible mechanism in peer-to-peer networking. Network coding is an alternative way to fragment files that can have much improved performance for large files in disrupted networks. ENCODERS adds both of these capabilities to Haggle, and they are implemented as standalone managers, namely the Fragmentation Manager and the Network-coding Manager. This design allows them to be turned on or off depending on configuration and, dynamically, based on the type of traffic, or other features of the situation.

Network coding provides localized loss recovery and path diversity with moderate overhead [11,16]. In MANETs that have an abundance of path diversity, network coding produces linearly independent blocks, thus mitigating the block-transfer scheduling or piece-selection problem that occurs when only local information is given and nodes dynamically join/depart. Network coding also helps to increase the number of distinct blocks available in the network thus providing a higher chance for peers to pull useful blocks [1,2,16]. In a MANET environment, network coding can take advantage of the broadcast nature of transmissions as well as node mobility [1,2,16].

Fragmentation provides an alternative to network coding for splitting a large object into smaller segments to take advantage of multiple source distribution and partial object transmission. Compared to sending the content in one data object, fragmentation has an increased dissemination overhead (it must manage which segments to transfer and which segments a receiver already has). This information is already maintained by Haggle by means of Bloom filters and disseminated as part of the node descriptions (at least in a delayed fashion). However, fragmentation has less computational overhead than network coding, and is thus a good comparison point for performance evaluations. Furthermore, if the number of blocks needed for network coding becomes large, the header overhead for the coding coefficients can become significant. Hence, in practice it is necessary to combine fragmentation and network coding for transmitting a very large data object. We implemented this hybrid scheme in ENCODERS. The fragments are usually referred to as *generations* in the network-coding literature.

4.4.1 Network-Coding Approach

Network coding is known to be beneficial in wireless and mobile ad-hoc networks in various applications due to its utilization of multipath and partial transmissions. The basic concept is to perform coding operations among content blocks at each participating node. By receiving enough (perhaps not all) linearly independent coded blocks from any nodes, the receiver can recover the original content. Network coding has been proven to be sufficient to achieve the maximum capacity of a multicast session in a lossless network [3]. It is also a natural remedy for partial, unsuccessful transmissions over high-loss channels and thus improves wireless communication reliability. These benefits derive from the information-mixing nature of network coding. By mixing information within a generation of content blocks, each network-coded block has the same degree of fresh information, no matter which path or which encoder it is from. Thus, network coding solves many issues in multipath transmissions and efficiently exploits previously received partial data objects.

Our implementation is based on an adaptation of Code-Torrent [1] to content-based networking with content caching and interest-driven content dissemination. Figure 7 illustrates the main idea. Assume content (e.g., a file) is already cached in multiple nodes of the network (three sources in this scenario). Once the content is requested, i.e., the interest matches the content attributes, network coding is invoked to split the content into blocks (three in this example) and code them into a single block, which is a random linear combination. In general, each source sends a stream of such blocks. The random coefficients are always new so that with high probability the same block is never generated again and a receiver will receive new information (i.e., a block that is linearly independent of previous blocks) with high probability. Hence, in our example the receiver only needs three blocks to decode (solving a system of linear equations) and reconstruct the original file. It does not matter where these blocks come from, how they were routed, or if there was a temporary disruption. Unlike traditional fragmentation, each block is as good as any other and has additional information content.

4.4.2 Implementation of Network Coding and Fragmentation

Haggle is centered on the notion of a data object, which is composed of the attributes and the plain text or encrypted content of the objects. Our Network-coding Manager will not encode the attributes but only the potentially large content part of the data object. The attributes are not encoded because Haggle needs to resolve interest against data objects for the purpose of content-based forwarding to peers.

We overcame several challenges with our implementation. The Haggle framework and its components operate under the assumption that a data object is a valid and fully received object (there are no partial data objects). Haggle transmissions operate in a transaction mode where an object is sent and received in an all or none manner. Unmodified Haggle does not support partial transmissions, resuming fetching an object, or multipath transmissions, which are essential for network coding to be beneficial. Finally, the network-coding protocol exchanges coded blocks, so there must be a mechanism to store and forward these blocks at intermediate nodes.

Our design represents blocks, which are generated by the encoder, as special data objects. Block data objects are created on demand when a normal data objects needs to be sent, without being inserted into the data store at the sender node (and thus avoiding unnecessary data base overhead). At the receiver and potential intermediate nodes, however, they are treated as normal data objects — that is, they enter the data store until they are purged, as they may be when sufficiently many innovative blocks have been received to recreate the original data object. This design allows us to experiment with forwarding of network-coded blocks through intermediate nodes and allows potential receivers to collect blocks from multiple sources, which is essential to exploit the benefits of network coding in mobile networks.

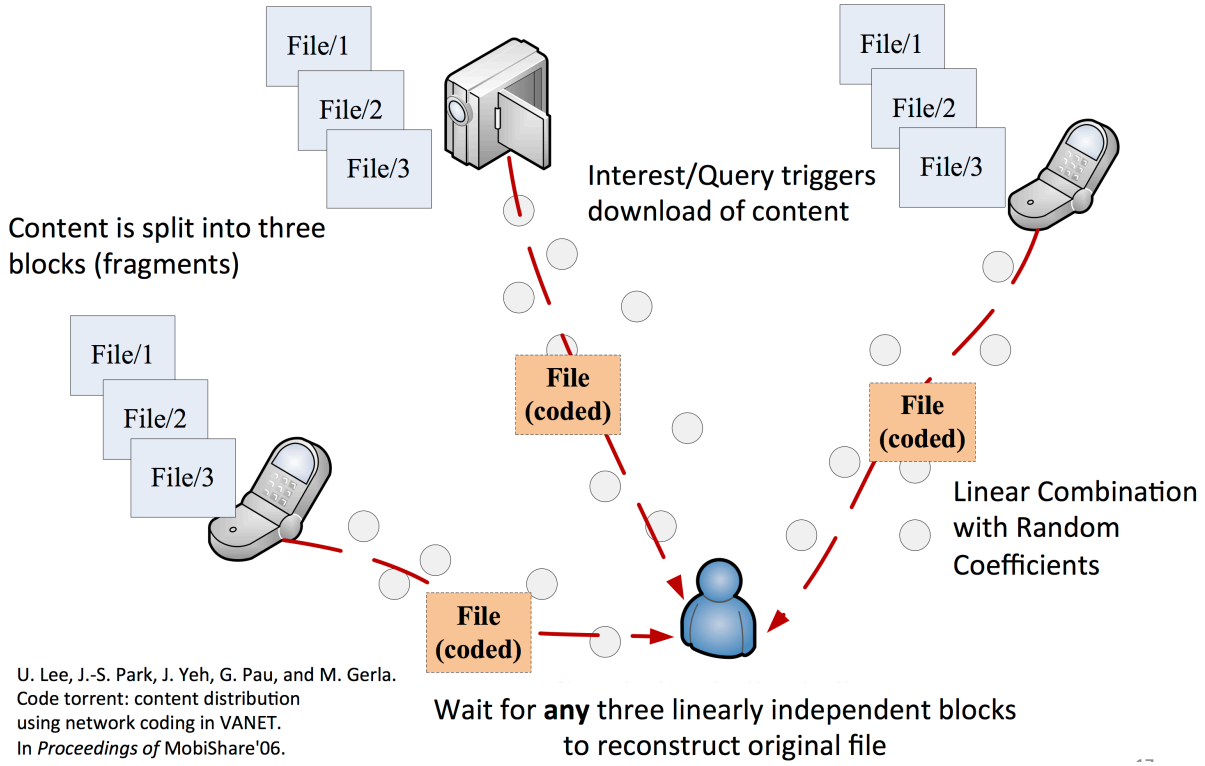


Figure 7. Network Coding. Each of the 3 devices is publishing the same file, and the Network-coding Manager on each device encodes the file into 3 blocks. If any 3 of these 9 blocks are received by the user, ENCODERS can reconstruct the file. Using Fragmentation, the user would have to receive all fragmented blocks from one device.

Our design allows network coding to be turned on or off depending on configuration and dynamically based on the type of traffic. Haggle natively supports associating attributes with a data object, which we leverage. Block data objects inherit all attributes of the data object being encoded, as well as several additional attributes. See the Technical Documentation, which also describes the classes, events and event flow for network coding and fragmentation.

Fragmenting files is an indispensable mechanism in peer-to-peer networking. Although the content-based approach and the interaction between fragmentation and the other

ENCODERS features is interesting by itself, we focus on the differences in implementation of fragmentation versus network coding in ENCODERS. Fragmentation closely follows the network-coding event flow given in the Technical Documentation.

Due to the receiver needing every unique segment to reconstruct the parent object, we call our approach *informed randomized fragmentation*. Our design leverages the Bloom-filter knowledge exchange to determine which segments the receiver already has, so that the sender can send only the missing fragments. The missing fragments are shuffled at the sender so as to increase the diversity in a MANET environment where there may be multiple concurrently contributing sources for a single data object. Randomization also helps if the content of the Bloom filter is not fully up-to-date, as is typical in these environments.

4.4.3 Context-Aware Network Coding Implementation

In Phase 2, we implemented the Context Aware extension of the network-coding algorithm (CA-NetCode), which adaptively switches between network coding, low-overhead data object fragmentation, and atomic data-object transmission. Depending on the context, CA-NetCode automatically turns network coding on and off for a given data object and target node pair. The objective is to eliminate unnecessary header and processing overhead when network coding is unlikely to improve the performance, but trigger NetCode instantly when it is most needed, by using simple context indicators.

CA-NetCode monitors the link-loss rate and maintains a loss-rate estimate for all known one-hop neighbors. The estimation of loss rate is a weighted moving average over a period of time. The loss rate is estimated based on the number of successful receptions of periodic beacons at the link layer. We use the following equations to compute the loss rate estimate:

$$\alpha = 1 - e^{-\frac{\Delta t}{W}}$$

$$l_n(t) = \alpha \cdot b_i + (1 - \alpha) \cdot l_n(t - 1)$$

Where W is defined as the size of the moving window, Δt is the duration between beacons, and b_i is defined as 1 if no beacon was received (i.e. there was a loss) and 0 otherwise. $l_n(0)$ is a configurable parameter; a value of 0 would fit with the assumption that connectivity is excellent until known otherwise; while a value of 1 would fit with the assumption that connectivity is poor until known otherwise.

By default, network coding is disabled for new data object and target nodes. At the sender side (i.e., the sender can be a forwarder or a data source), network coding is enabled for data objects satisfying the following two conditions:

$$P_n(t) > P_{thres}$$

and

$$S(d) > S_{thres}$$

where P_{thres} and S_{thres} are the thresholds of link-loss rate and data-object size, respectively, $P_n(t)$ is the average link loss rate at time t for a target next hop node n, and

S(d) is the size of data object *d*. For data objects with size larger than S_{thres} that are not suitable for network coding, the data object is fragmented at transmission to reduce transmission failures due to large size. In our implementation, the link-loss rate is estimated as a moving average by observing the periodic beacons at the link layer.

At the receiver side (i.e., the receiver may be a relay node or the intended receiver), if a network-coding block is received from any of its neighbors, the previously received un-encoded fragments are converted to encoded blocks. For a data object composed by *m* blocks/fragments, and I_j is the unit vector whose *j*-th element is 1, the *j*-th fragment f_j of a data object is converted to an encoded block b_j by

$$\boldsymbol{b_j = I_j[0_1, 0_2, \dots 0_{j-1}, f_j, 0_{j+1}, \dots, 0_m]}$$

In this way, all received fragments and encoded blocks can be used for decoding the original data object so that all successful transmissions are utilized.

4.4.4 Limitations and Possible Future Directions

Efficiency of Network Coding. Unmodified Haggle supports a single data-object container without support for fragmentation or blocks, which makes it difficult to support multipath transmissions, resuming of transmissions, and efficient forwarding for network coding. SRI's solution represented blocks as ordinary data objects, which can lead to performance issues because each data object is stored and forwarded individually. Aggregation at intermediate nodes might help, which would also enable the mixing of blocks that have been aggregated. Furthermore, for the first iteration, our design used a single generation for an object and a fixed generation size, which limited the content size to which we can apply network coding. This can be mitigated by fragmenting large data objects into multiple generations, which can be transferred concurrently. Future work could explore variable generation size, and the rate of generating block data objects for better information diversity and performance. To improve reliability, one could study how the network-coding parameters affect delivery of blocks. The performance of network coding needs to be studied in the context of unreliable protocols and broadcasting, e.g., utilizing UDP or NORM.

Countermeasures against Pollution Attacks. Because Haggle creates a hash signature composed of the data object's attributes and contents, pollution attacks are possible if the relays are able to mix packets. To reduce the vulnerability to such attacks, our design dictates that only the source node will encode the blocks. Relays simply forward block data objects from their local store. Future work may examine the tradeoffs of allowing the relays to mix already encoded block data objects. Recent research on homomorphic signatures for network coding provides some solutions, but the computational overhead is significant. Results need to be carefully examined to understand if the techniques are suitable for our resource-constrained Android platform. Less expensive alternatives that are based on identifying the source of pollution and taking suitable countermeasures seem preferable. Our current approach uses relatively large block sizes (e.g., 32K), which not only leads to a better amortization for content-processing overhead, but has the advantage that the number of blocks is generally small enough so that they can be individually signed and verified with reasonable overhead.

Reusing Computations. Images may be edited and transcoded for multiple screen sizes, which can be see as another form of coding on top of a network-coding protocol. This raises the question of how to efficiently transmit, fetch, and store these data objects. For instance, if an image of a map is edited, the resulting content may be the original image, the edited image for the screen size it was edited on, and the "*diff*" that describes the transformation applied to the image. Nodes have the option of retrieving the edited image and transcoding, or fetching the original image and the diff and transcoding for their screen size. Finally, this resulting image is stored as content as the transcoded image and the diff. This allows us to take advantage of network coding's linear independent segments, and reduce the necessary content needed to fetch the latest version of an image for a particular screen size. It might be worthwhile to investigate if significant bandwidth reductions are possible using such an approach, which moves the idea of network coding to a higher level of abstraction.

4.5 Security

4.5.1 Overview

Figure 8 depicts the scope of the security solutions developed by SRI in ENCODERS within the context of the threats, features, and existing security solutions.

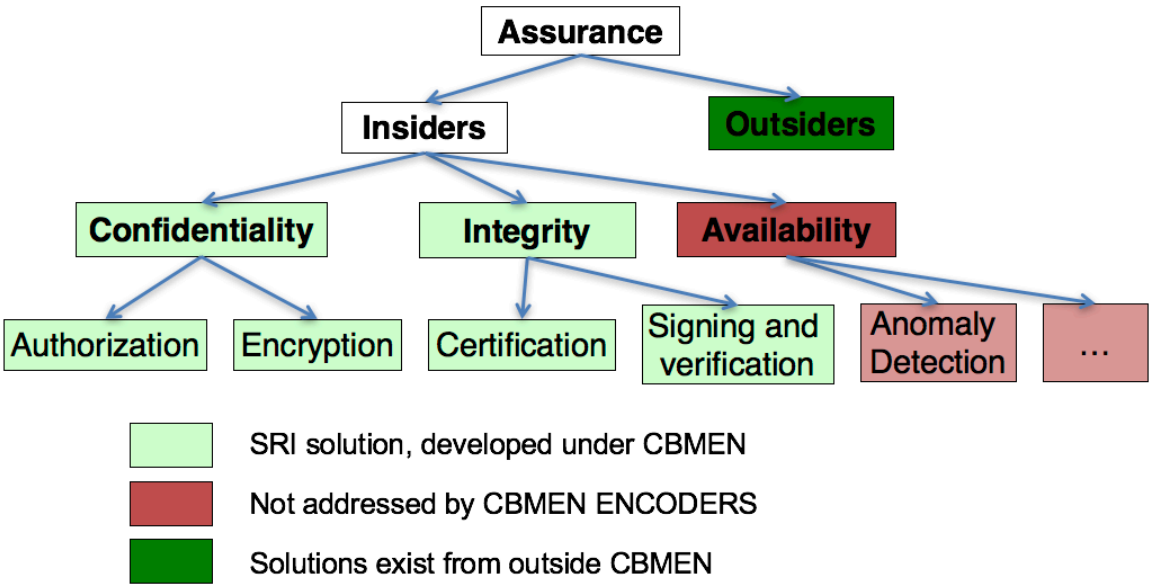


Figure 8. The security solutions developed in ENCODERS for confidentiality and integrity are shown in the context of the broader assurance tree.

ENCODERS' security is layered atop extant assurance. Commodity devices support link-layer security using symmetric cryptographic primitives. We assume such security is utilized by all *insiders* – that is, legitimate participants – and that they all share a cryptographic key (typically derived from the password needed to join the network). All others are referred to as *outsiders*, and can be distinguished by the fact that they do not possess this key.

Because current radio-level protection is coarse-grained, only protecting against outsiders, ENCODERS focuses on mitigating threats from insiders. Ensuring the security of content requires that its confidentiality, integrity, and availability are all assured. The integrity of content is ensured by signing it before it leaves a node, and verifying it upon arrival at another node. Similarly, confidentiality is provided by encrypting content that leaves a node, such that only suitably authorized nodes can decrypt it. Availability is not ensured in ENCODERS. A future extension could combine node-specific accounting with distributed attribution to effect decentralized resource management (and thereby manage availability).

ENCODERS provides flexible trust configuration by supporting arbitrary nodes being configured as certification or authorization authorities that are responsible for providing cryptographic key material to insiders. Authorization can be specified using roles to simplify configuration.

4.5.2 Approach

The confidentiality and integrity of content are particularly crucial in peer-to-peer contexts where some participants may not be trustworthy. The ENCODERS security design provides substantial value over the traditional approach, automating the process of ensuring that information is accessible to authorized principals and not to others, leaving the warfighter to concentrate on other things.

In our design, the publisher of content specifies the *access policy* for that content in simple terms based on authorized *attributes* of the recipient. Content is encrypted when it leaves the publishing node, may be forwarded by nodes without authorization to decrypt, and is eventually decrypted at nodes with the right attributes. Our design uses attribute-based encryption, which allows us to cryptographically implement complex security policies. Applications can dynamically set access control policies that are specialized according to the content and context. This provides access to those that are authorized and denying access to all others, with a reduced negative impact on latency and computational overhead when publishing and retrieving information.

This design description is focused on the key aspects of ENCODERS security, namely:

1. Integrity and non-repudiation.
2. Trust Establishment.
3. Confidentiality.

(The Technical Documentation gives details of the security implementation, including background signing, decentralized certification, lazy encryption, cached decryption, key distribution, and signature chaining.)

Integrity and Non-Repudiation. ENCODERS creates a signing and verification key pair per node. When content leaves a node, its hash is signed. The input to the hash includes all the descriptive attributes as well as the content itself. Signature verification is a check that the content came from the node in the system that it is claimed to be from. Because node descriptions are sent frequently, they can optionally be sent without signatures to reduce load (see sign_node_descriptions in the Configuration Manual).

Signature verification of a data object requires that the receiver have the sender's public key. This may not hold if the two nodes have never met, or if the receiver does not trust the sender directly. To allow data objects to flow in such a case, we introduce *signature chaining* as a configuration option. With this functionality, each node can re-sign the data object before forwarding it, adding its signature to a list. A subscriber only has to *directly* trust the last sender, and can decide whether to implicitly transitively trust the chain of signers along the path from the publisher, or to explicitly examine the list to make a choice.

Decentralized Certification. In unmodified Haggle, the same fixed certificate-authority public and private keys were provisioned at all nodes. This was used to sign the public key for each node and to generate a corresponding certificate. These certificates are exchanged in node descriptions during encounters between devices. Because the same certificate-authority key pair was present on all nodes, a node would accept the certificate of any other node and thereafter trust it. Accepting all certificates effectively means that the signing mechanism did not provide any security guarantees.

This problem was addressed in ENCODERS by introducing a certification framework, where authorities each possess distinct Certificate Authority (CA) keys that are used to certify the public keys of other nodes. For this purpose, we distinguish between nodes that are *authorities* versus those that are *users*. User A can get its public key signed by multiple authorities (and hence have multiple certificates). User B will trust user A if it trusts any of the authorities that have signed a certificate for A. Any node can act as an authority, by adding an “Authority” section in the configuration.

An authority node will listen to requests for certificate signatures and respond to them appropriately, returning the signed certificate along with the authority’s own certificate. The authority node will maintain a list of nodes that it will issue signed certificates for. This will be used to restrict network access to only authorized nodes. To ensure their confidentiality and integrity, requests and responses are hashed and encrypted with a symmetric key (specified in a node’s configuration file) that has previously been shared out-of-band between the authority and the user.

Trust is established using a transitive relationship, as mentioned previously. When node A meets node B, there will be an exchange of certificates, with each node sending all its certificates to the other. Node A will verify each certificate from node B, by looking for the appropriate authority certificate in its trusted authority certificate set, and using it to verify the certificate from node B. If any of the certificates from B is successfully verified, node A will trust node B and use the public key to verify objects from node B. If none of the certificates are trusted, node A will store them in a set of unverified certificates.

A user node maintains a list of authorities that it trusts, along with a list of shared secrets to use for communication with each authority. It will periodically broadcast its public key, waiting for authorities to sign it and return a certificate; stopping once it has received all signed certificates. It will save the authority’s certificate in its set of trusted

authority certificates, and will proactively extend its set of trusted nodes by trying to verify all unverified certificates using this authority certificate.

Confidentiality. Unmodified Haggle does not protect the confidentiality of content. However, access control must not become a single point of failure or limit network scalability. This precludes centralized solutions, such as the use of traditional asymmetric ciphers that require the retrieval of the recipient's public key. While remote directory lookups can be avoided through the use of identity-based encryption [9], this allows a sender to generate the recipient's public key locally, and policy-based data sharing among a group would still require an alternate solution to eliminate the use of a trusted server (for group management and policy enforcement).

Our solution uses a variant of *ciphertext-policy attribute-based encryption* [12], which lets a system-wide authority give each user decryption keys with appropriate attributes embedded. When content is encrypted, the sender defines the access policy in terms of the attributes that are needed to decrypt the content. The policy is a tree with operators (such as conjunction and disjunction) at internal nodes and attributes at the leaves, allowing very expressive authorization. Because the policy is embedded in the content-encryption process, enforcement is purely cryptographic and provides an end-to-end guarantee with no intermediate service needed to mediate access.

Cryptographic keys are typically certified hierarchically, with trust flowing statically from a single authority. In the past, a coordination phase was required to enable interoperation between groups from different trust domains. *Multi-authority attribute-based encryption (MA-ABE)* [4] allows content to be encrypted with a policy framed over attributes from different authorities without any prior communication. We use MA-ABE to construct cryptographic capabilities for content, allowing trust to be managed orthogonally from key certification. This allows trust to be reconfigured locally, when the network is partitioned, by coordinated adjustment of the set of acceptable attributes.

When a user node first learns of the existence of a given authority node, which is after receiving a signed certificate from it, it will eagerly request all the cryptographic attributes that it is authorized for. The authority will respond appropriately. When a user node uses a policy for which it does not have the required cryptographic attributes, it will request those attributes from the relevant authorities. It will periodically repeat those requests until a response is received.

Access policies can be any Boolean formula over the attributes. Specifically, authority and attribute identifiers must be alphanumeric. The authority whose namespace is used to define an attribute must always be specified, with the two separated by a period. Thus, an attribute *Attr* from authority *Auth* is specified by *Auth.Attr* and is an atom in a policy. Attributes can be connected with *OR* and *AND* operators. Parentheses can be used to indicate precedence. The following is an example of a valid policy:

Authority1.Attribute1 OR
(Authority2.Attribute1 AND (Authority3.Attribute2 OR Authority3.Attribute9))

4.5.3 Using Access-Control Policies

Our design of limiting access based on attributes provides a flexible framework. ENCODERS provides a role-based access control (RBAC). Each authority specifics a list of role names, which are appropriately scoped by the issuing authority's name. Thus, a role *Role* from authority *Auth* would be named *Auth.Role*. The authority specifies the sets of encryption and decryption attributes for which a given role is authorized.

For each role, the authority also specifies a shared secret. A user node will specify a list of roles, along with associated shared secrets. When a user node makes requests for attributes, it sends the list of role names as part of the request. The authority will respond with the encryption and decryption attributes that each role is authorized for, with each encryption/decryption attribute being encrypted with the role's shared secret. The user node will decrypt the encryption/decryption attributes using the shared secrets. Note that this allows for monotonically increasing the access at any given node, by adding more role shared secrets. These can be added through the use of a control message, without requiring a reboot.

The role based access mechanism allows for the decoupling of a user's authorization from the device's authorization. Devices are initially provisioned with a minimal set of roles, and their access can be upgraded based on the requirements of the users using the given device. In situations that require it (such as device failure), a user can simply move to another device, enter the appropriate shared secrets, and have access to data that they are authorized for on the other device.

Traditional access control, where permissions are granted to specific subject-object combinations, can be implemented by using an access policy for the data *object* that requires the receiver have a key with the *subject* attribute. The use of groups provides syntactic sugar for easier definition of access policies. Group-based access control can be implemented by encrypting the data *object* with an access policy that requires the receiver have the appropriate *group* attribute.

As part of our implementation, we extended the Haggle Security Manager. The desired policy is added as an attribute of the data object being published. The Security Manager uses the specified access policy to encrypt a key (used to symmetrically encrypt the content) with MA-ABE. The data object is routed from the publisher to the subscriber based on the recipient's interests. If the receiving node has the appropriate cryptographic attributes, it can decrypt the content.

4.5.4 Concept of Operations (CONOPS) for Security

This section describes the concept of operations (CONOPS) for the ENCODERS security solution for small-unit operations of warfighters at the edge. Examples of applying the ENCODERS security design to different operational scenarios are given in the Security CONOPS section of the ENCODERS Software Design Description.

The ENCODERS security solution provides authentication, non-repudiation and integrity by the certification and signing/verification mechanism. Access control is provided by Multi-authority attribute-based encryption of content transmitted through the network. Encryption attributes can be grouped in roles to simplify adapting to dynamic change of mission or personnel.

ENCODERS can be configured for different levels of security with authentication and access control configured independently. There are *three authentication security levels*: no signing (LOW), only node descriptions are signed (MEDIUM), or all data objects (metadata plus content) are signed (HIGH). Encryption can be enabled or disabled. Even if encryption is enabled, it is the application's choice as to encryption policy, which could be none, a policy that gives access to all nodes in the ENCODERS network, or policies that are fine grained.

ENCODERS security is *designed to be invisible to the warfighter*. The key decisions are made at design time, when the security and interest attributes are defined and security attributes are assigned to warfighters. Assignment of security attributes can provide the equivalent of traditional MLS security, well as expressing need-to-know properties.

It is expected that applications will be configurable to assign correct access-control policies to content that they generate. It is also possible for an application to expose policy decisions to the warfighter. These are choices that need to be made by leaders who understand the mission security needs and have the knowledge and authority to make such decisions.

The ENCODERS API is essentially the Haggle API, which is implemented in the libhaggle library (http://www.haggleproject.org/). Documentation of the API is included in the ENCODERS Technical Documentation. A key decision in designing security is setting a policy attribute on data objects that contain content to be published. Additional capabilities are provided for dynamically configuring security attributes and roles. This is explained in the ENCODERS Software Design Description.

The ENCODERS package includes a security configuration tool, discussed below, to simplify the configuration process once the security space has been designed. Once configured, provisioning happens automatically as part of the ENCODERS startup process. The Software Design Description describes the details of pre-mission security design and configuration.

Security issues during mission execution. We assume that the device assigned to a warfighter (WF) locks when not in use, and requires a PIN (hopefully known only to the WF) or thumb print, or other biometric access information to unlock. We discuss the security aspects of ENCODERS devices for the following situations:

- Encryption of data in device
- Device requires a reboot -- hangs, battery falls out
- Device needs to be replaced
- Warfighter role is changed
- An unauthorized user accesses the network using an ENCODERS-enabled device
- Device operating as Data Couriers or Delegate Nodes

Encryption of data in ENCODERS devices. Data that is published with an access-control policy is encrypted when transmitted to remote nodes. It also resides locally in whatever form the publishing application produced. Encrypted data arriving from remote nodes will be decrypted if the receiver has security attributes that satisfy the access policy, before delivery to interested local nodes. Data encrypted with a policy not satisfied by a node's security attributes will remain encrypted.

Keeping all data encrypted does not help. The node will have the decryption key and so anyone getting control can decrypt. Further more, the data is created by and given to applications in the clear, and ENCODERS has no control over what the applications do.

ENCODERS device requires a reboot -- device hangs, battery falls out. ENCODERS will reboot using persistent state. Thus only tasks in progress and recent (non-persistent) state changes will be lost. The long-term security state consists of the trust base (certificates of trusted nodes) and the node's security attributes (and associated keys). These are kept in the datastore (an SQLite database). If the database is kept on disk, then only updates in progress will be lost. For better performance, the system can be configured to use an in-memory database, which is persistent after the initial startup phase, and periodically updated during operation.

In a controlled shutdown, the in-memory database is written to a persistent store. To protect against uncontrolled shutdown, the system can be configured to backup the in-memory database at a specified frequency. Security tokens that were acquired dynamically and recently may not be persistent, but they can be reacquired dynamically.

Tasks residing in event queues will also be lost, which means some work will need to be restarted. The main loss would be content published by local applications that has not been cached in the local datastore or transmitted to the other nodes.

ENCODERS device needs to be replaced, for example broken or lost. It may be possible that there are spare devices that have been initialized at the FOB. Such a device would have its signing certificates, and all the roles in encrypted form. The warfighter can use his access key to unlock roles for which he is allowed access.

If there are no spares, a warfighter might commandeer a device from a lower-level warfighter, and unlock any allowed roles that are still encrypted.

Warfighter role is changed. This is similar to the above case where there are no spare devices. The warfighter may have been entrusted with the password or may obtain it from a higher authority as needed.

Unauthorized user accesses the network. We assume the user has an ENCODERS-enabled device. Nevertheless, the user may not be part of the mission network. The ENCODERS trusted-set signature verification and access control will prevent this user from injecting or seeing content.

ENCODERS devices operating as Data Couriers or Delegate Nodes. These devices will be part of the trusted set in the sense of signature verification. They will not have access to any decryption keys and thus cannot deliver any decrypted content to applications on the node. They subscribe to content that they should carry or forward.

The ENCODERS Software Design Description describes mission-specific aspects of our security CONOPS: pre-mission security design and configuration; and examples of security designs for multiple scenarios for small-unit operations of warfighters at the edge.

5 Evaluation

Simulation/emulation is key for early design testing, self assessment, and scalability evaluation of a highly configurable design such as ENCODERS. For effective modeling of CBMEN information flows, the test framework needs to: (1) allow repeatable execution of scenarios with different patterns of network traffic operating in different military contexts, (2) provide a rich simulated/emulated environment that can model virtually any network topology and mobility, and has high-fidelity models of military radios, and (3) support flexible large-sale simulation, with the option of using, for some of all of the nodes, virtual machines that execute the same code that is or would be used on an actual device.

The DARPA CBMEN Program Manager suggested that a systematic parameter-space exploration would be an important outcome of the CBMEN program. The SRI team invested a lot of effort to develop suitable tools to automate this process. The result is a fully automatic test generation and execution framework that can run a large number of simulations using CORE/EMANE in resource-constrained Linux containers that approximate the performance of the CBMEN target platform. Experiments with up to 70 nodes are currently possible and 30 to 50-node evaluations were routinely performed by the SRI team, mostly in overnight runs on a number of dedicated servers. The reproducability of the results is good, because of the concise specification of scenario parameters that our tools take as input. While the quantitative evaluation has been mainly conducted using this Linux-based test framework, the technology testing and evaluation has been complemented by a large number of tests on a 30-node Android phone testbed at SRI.

Early in Phase 1, we proposed the following Value Proposition, and this proposition has informed the tradeoffs made in our design. In the context of a tactical-edge MANET, which includes network disruption and reconnection; and limitations on bandwidth, range, transmission power, computing power, memory, we expected ENCODERS as proposed to:

- Deliver the highest-priority (most relevant) information available to the warfighter in a timely manner.
- Limit or eliminate the delivery of non-relevant information.
- Ensure that information is provided only to those personnel with authorized access, but without the inconvenience or overhead of key management.
- Provide an additional layer of more fine-grained security and trust by operating on top of existing, possibly certified, security solutions.

We assert this proposition was achieved and in this section we present our evaluation results to support this claim. The achievement of the last two security-related bullets was described in Section 4.5. We measure throughput, delivery fraction (in terms of relevant data objects), and latency as system-performance metrics to support the first two bullets. In addition, to show that ENCODERS is a robust solution that can work

continuously, we present results for measures such as bandwidth utilization, power consumption, computational power utilization, and memory utilization. These are all essentially measures of system efficiency, which is important in the target domain of CBMEN. Finally, we also show the effects on latency of different security options.

Evaluating our approach of building on the Haggle framework: our experience was positive. We identified and documented some limitations and quirks, and found and fixed a number of bugs. However, we had few issues during the development of ENCODERS, and our developers were able to quickly master the complexity of the framework and the new components. This was largely due to the quality of the existing Haggle documentation and frequent SRI-led team discussions during Phase 1. The three documents of ENCODERS that are described in Section 1 incorporate the lessons learned and should be a valuable resource for those using or building on ENCODERS.

5.1 Methodology and Robustness Results

The use of the Linux version of Haggle for rapid prototyping and testing in CORE Linux containers saved development time, because testing on Android phones is significantly more time-consuming.

SRI made modifications to Haggle to significantly improve its performance, which enabled us to run 30-node scenarios in Phase 1 on a single machine with CORE and Linux containers. The performance improvements were needed for a component-level performance evaluation for networks of this size. SRI designed a set of scripted scenarios for CORE/EMANE that were extended as features were added.

In parallel with the performance evaluation on CORE, SRI developed a testbed for 30 Android phones. We conducted over-the-air tests on SAIC test cases to better understand the limitations of the CORE/EMANE models, in particular in regard to channel contention and packet losses.

As a further aid to evaluation, SRI developed its own scripted tests using a small test application that exercises various API features. Running these tests was useful to stress the system to find subtle timing or multi-threading problems that do not show up with Linux containers on CORE.

Our test framework was further extended to run automated parameter-space exploration studies and regression tests, based on concise scenario parameter specifications. The framework is based on CORE/EMANE with Linux containers and enables us to run a large number of tests on a set of Linux servers, typically overnight. We evaluate ENCODERS performance via highly parameterized tests in a repeatable way. Figure 9 shows the flow of our test automation framework, which is composed of (i) test generation and (ii) regression testing and evaluation. A user specifies a json file containing multiple test specifications (i.e., mobility parameters, application parameters, number of nodes, and so forth). *Test Generator* creates directories containing test cases that are compatible with *Test Runner*. The test results are logged in the form of graphs and tables for further analysis.

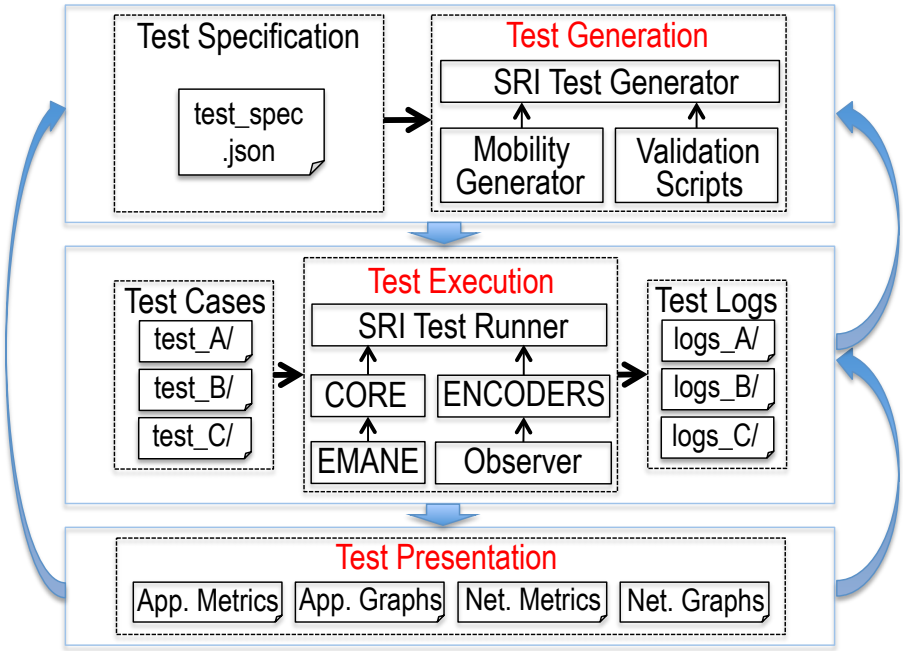


Figure 9. SRI's Test Automation Framework. A concise specification of scenario parameters is used to automatically generate tests (using the SRI Test Generator module) with the mobility of the nodes automatically detailed. These tests, which can be rerun reproducibly using the SRI Test Runner module, are then automatically run and results generated for human analysis.

During code hardening in Phase 2, SRI developed an *observer* mechanism to monitor the current status of node, network, and performance metrics. Basic observable information is collected to analyze and further tune the operation of ENCODERS. Figure 10 through Figure 14 show some available observables for the purpose of evaluating the robustness of ENCODERS after Phase 2. These figures show observables used for monitoring the resource usage (e.g., bandwidth, CPU, memory) and network status (e.g., connectivity measure based on the number of neighbors) from the platoon-level scenario (Section 5.3.1). The point here is that these graphs all show that the quantity in question (bandwidth consumption, CPU, and so forth) level off as time elapses, and do not exhibit undesirable exponential properties, providing evidence of ongoing, robust execution of ENCODERS at each node.

Other observables, not mentioned here, were also continuously monitored for self-assessment. These include observables on information flow, such as content and interest list. Data objects that are subject to dissemination can be observed with distributed monitoring, and the data can be further analyzed to optimize use of network resources given the priority and amount of information for transmission/dissemination.

Legend for observables graphs. The X-axis is time in seconds. Y-axis is the value of the observable in question, and each colored line represents the value of the observable over time for one node in the network. This is a 30-node scenario, so there are 30 such lines. Values for *total transmitted bytes (Tx)* and *total received bytes (Rx)*

include all traffic that occurs on the channel (i.e., both content and overhead used by ENCODERS to manage the network and deliver content).

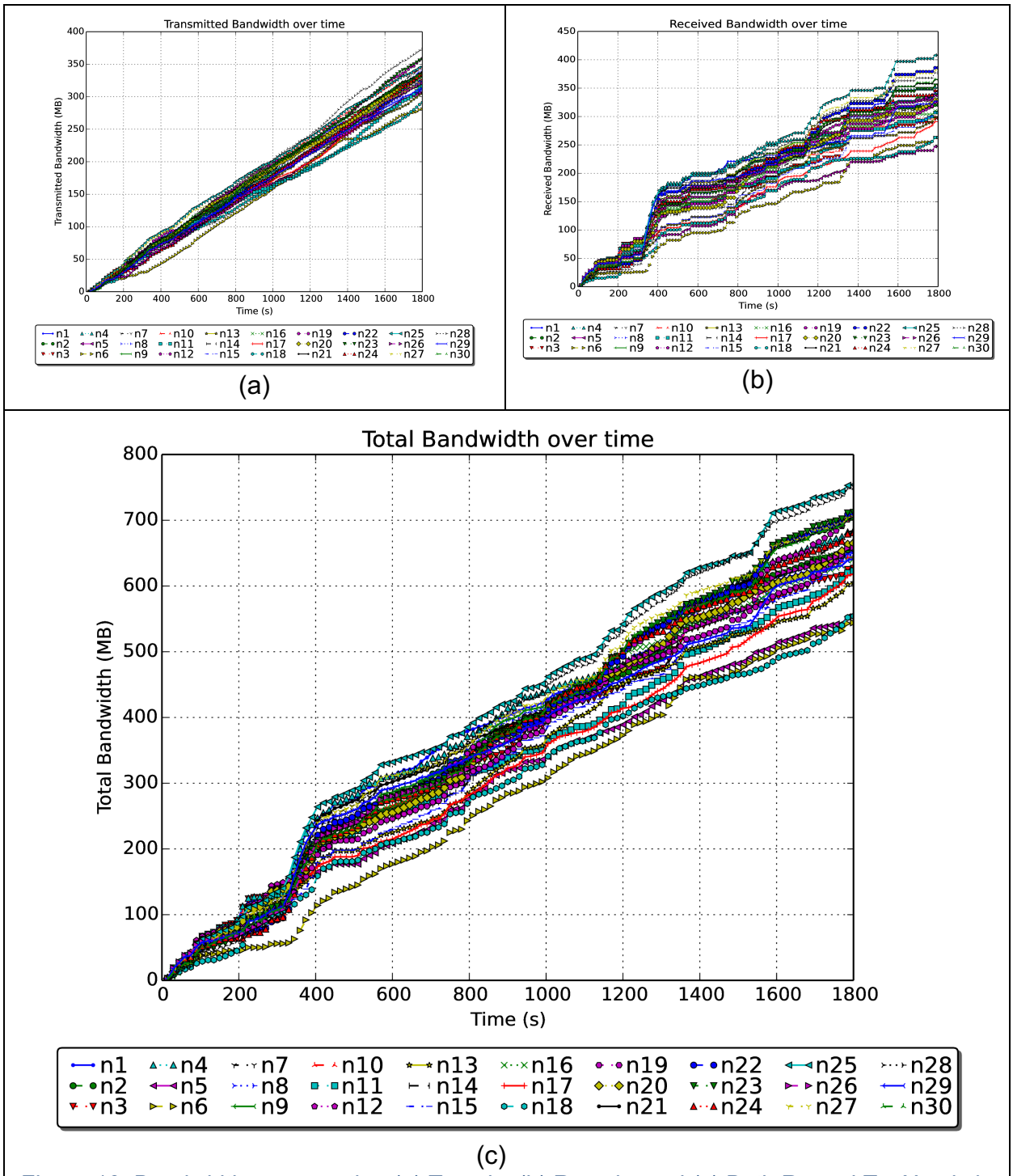


Figure 10. Bandwidth consumption (a) Tx only, (b) Rx only, and (c) Both Rx and Tx. X-axis is time in seconds. Y-axis is bandwidth used; each colored line is one node. The graphs do not exhibit undesirable exponential properties, providing evidence of robust, ongoing, execution.

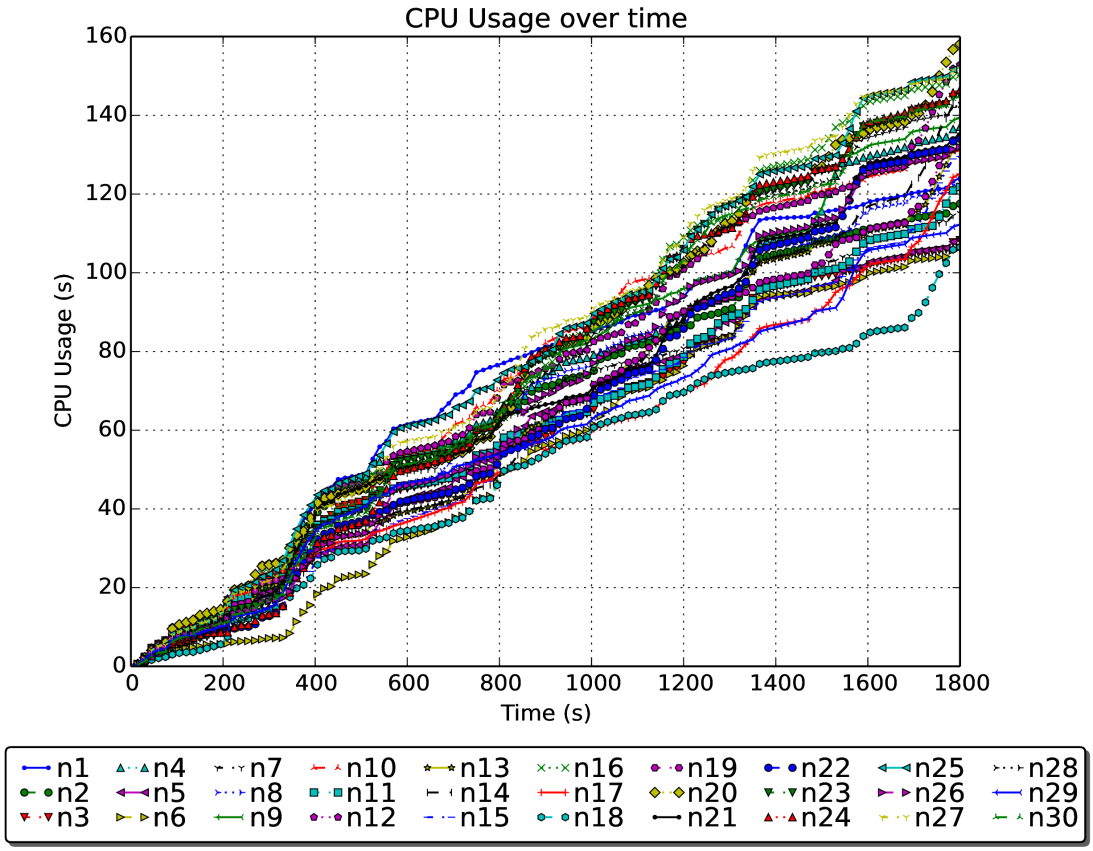
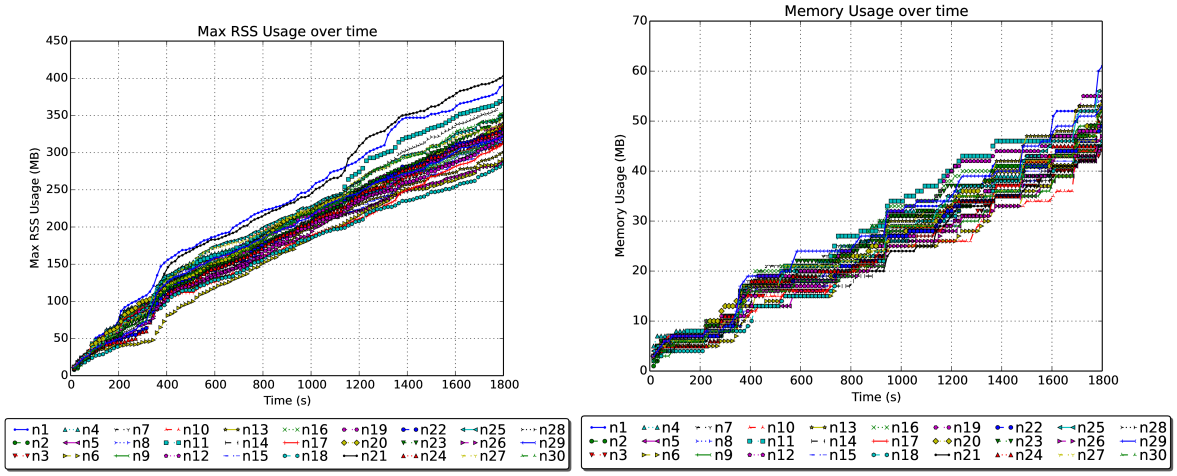


Figure 11. Cumulative CPU usage. The graph does not exhibit exponential properties.



(a) (b)

Figure 12. Cumulative memory consumption. The graphs do not exhibit undesirable exponential properties.

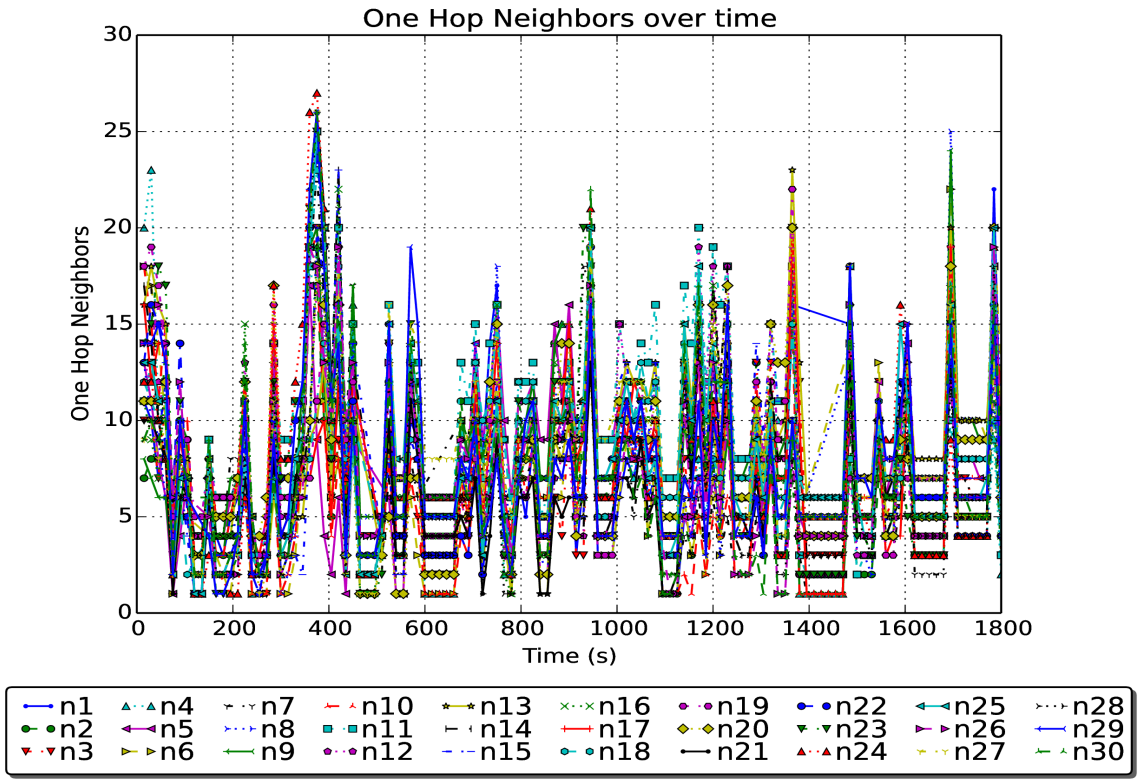
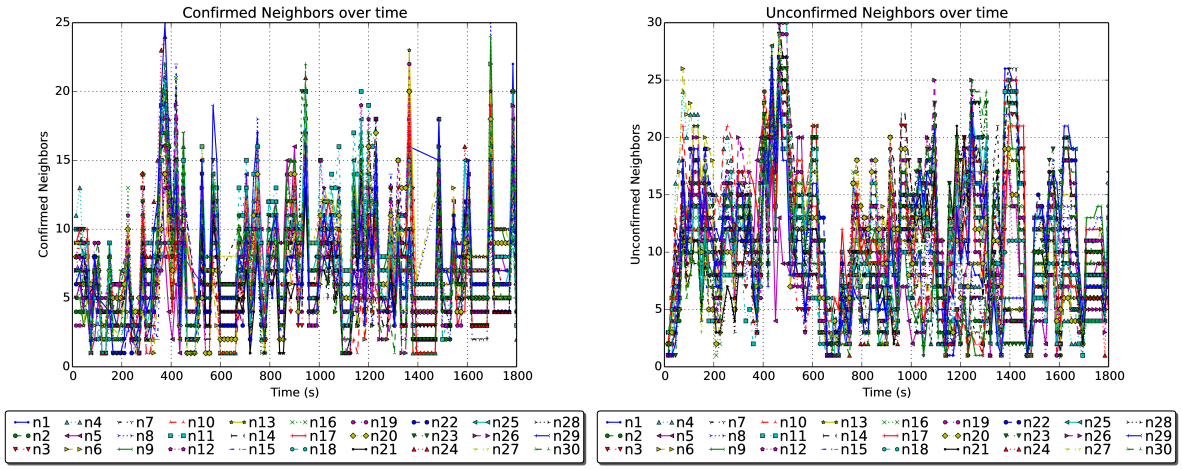


Figure 13. Number of physical neighbors at any given time. There are sudden spikes (e.g., before t=400) when groups of nodes come into contact with each other, such as when two or three squads of soldiers come into contact.



(a) (b)

Figure 14. Number of the logical neighbors at any given time, (a) shows confirmed neighbors, (b) shows unconfirmed neighbors.

Cache Size. To understand what factors affect network performance, SRI observed the use of the cache in our ENCODERS network. These experiments are in the platoon scenario described below in Section 5.3.1, which also presents more results on the sensitivity of ENCODERS to cache size.

Figure 15 shows the ratio of cache usage over time when the cache size is limited to about ¾ of the typical observed cache usage (when the cache is unlimited) in our scenario. We can see that the cache is heavily utilized and close to capacity, which is further evidence of the importance of cache management to ENCODERS performance.

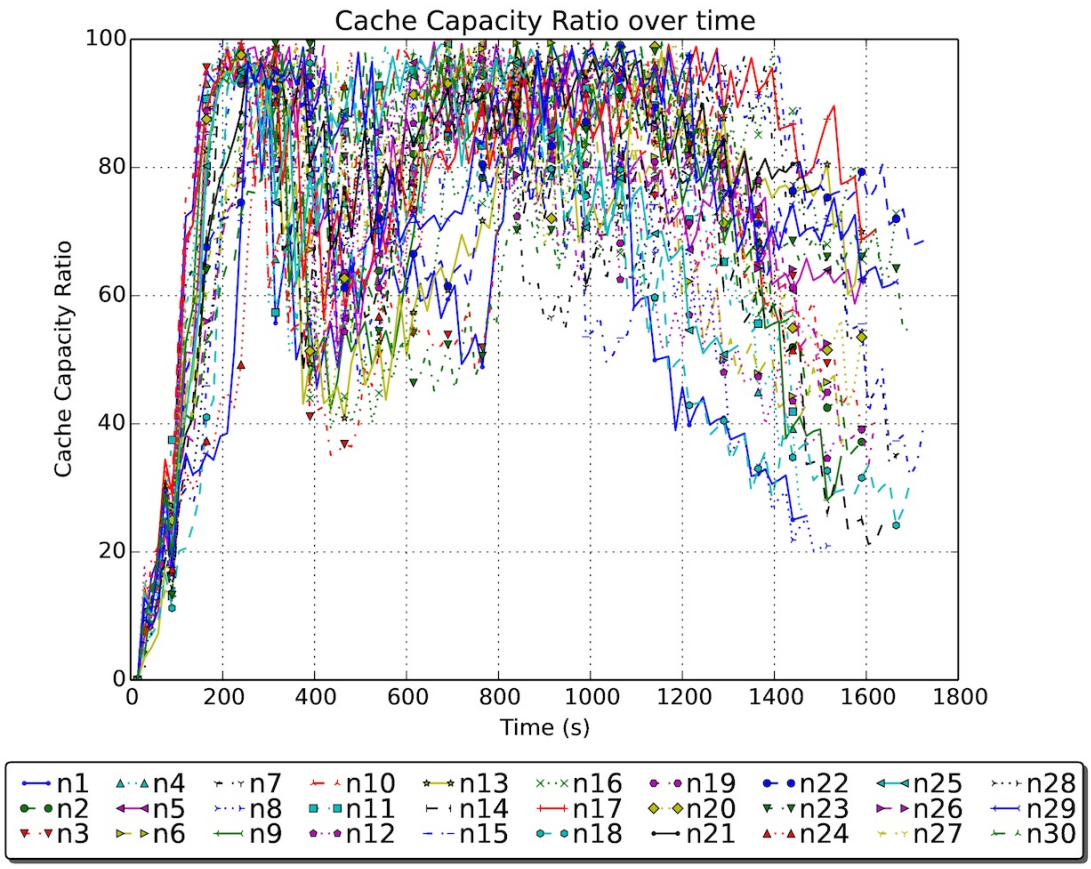


Figure 15. The percentage of cache capacity used by each node.

Eviction and Connectivity. Figure 18 shows the number of data objects that ENCODERS inserts and evicts in the course of managing the cache in our evaluation scenario. Evictions can be due to purging or replacement. Subfigures (a) and (b) show a fairly linear growth over time with small spikes, indicating that ENCODERS is managing the cache smoothly, consistent with long-term stability. Subfigure (c) shows the number of data objects hard evicted (dropped immediately on receipt, even before being insertion into the cache). By comparing to Figure 13, we see that hard evictions spike when connectivity spikes. Figure 13 showed a sudden spike in connectivity before t=400, when hard evictions also spiked. This happens when a new group of neighbors is added because there are now many cached items that are duplicated or outdated.

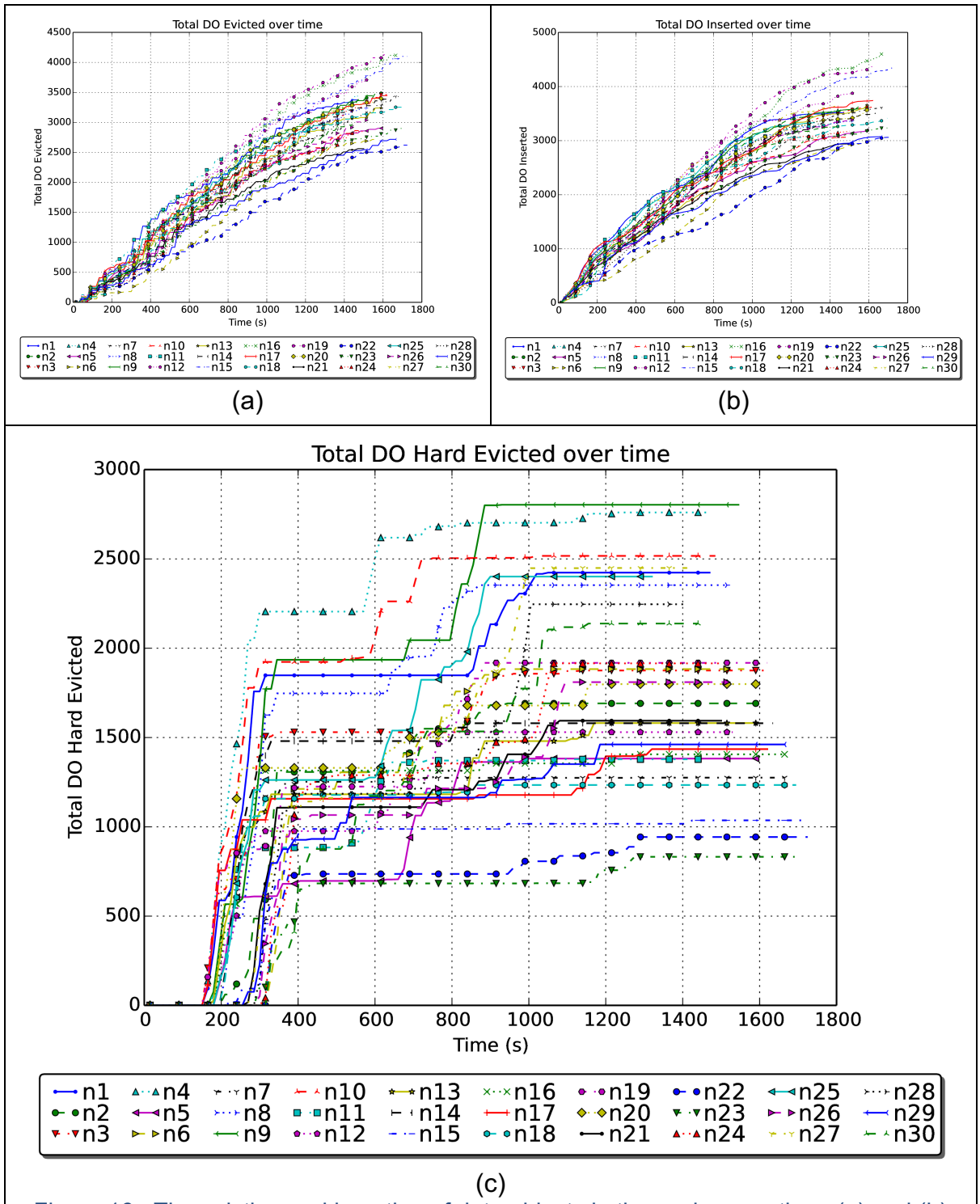


Figure 16. The eviction and insertion of data objects in the cache over time. (a) and (b) show total evictions and insertions and (c) shows hard evictions.

5.2 Summary of Phase 1 Evaluation Results

In Phase 1, we designed and implemented the overall ENCODERS concept and architecture to support (i) unified communication and search via content matching and ranking, (ii) universal caching and dissemination algorithms involving network coding, and (iii) decentralized access control using digital signatures and policy-based encryption.

In [19], we described (i) a complete ICEMAN architecture[3] that integrates multiple content-dissemination, utility-based caching, and transport mechanisms to provide a publish/subscribe API with attribute-based content naming, and (ii) content- and context-based policies that utilize these mechanisms to achieve efficient communication at the tactical edge. The design of our ENCODERS/ICEMAN architecture emphasizes compositionality in the sense that all features seamlessly interoperate with each other. Without architectural changes, our system supports any combination of the discussed caching, transport, and dissemination mechanisms.

SRI's Phase 1 evaluation of ENCODERS using our test automation system has been reported in [17]. This testing enabled us to understand the performance characteristics of different policies for controlling the network, by measuring latency, data throughput, and other metrics. We found that different dissemination, transport, and caching policies have significantly different performance characteristics (in terms of total data objects delivered and latency). A combination of content-based policies was necessary to achieve the best performance. The best observed performance was achieved with combinations of hard- and soft-constraint utility-based caching policies that rank data according to network context. They outperformed the use of only hard-constraint policies such as time-based purging and order-based replacement.

Phase 1 ENCODERS' field demonstration and evaluation results, primarily collected by other DARPA contractors, are presented in Section 3.

5.3 Phase 2 Evaluation Results

We evaluated the effectiveness of several policies for content transport, content dissemination, content caching, network coding, and security in ENCODERS. We used two scenarios for evaluation: (i) ground-based platoon-level operation with 30-50 nodes and (ii) airborne/ground network with satellite communication.

5.3.1 Ground-based platoon-level operations

Scenario. The applications and scenario model a 30-node tactical scenario with an explicit social hierarchy (three squads of 10 members each). To specify squads' movement, we use the Nomadic Community Mobility model in the BonnMotion scenario generator: http://net.cs.uni-bonn.de/wg/cs/applications/bonnmotion/. In this model, groups are performing a random walk around their reference points that follow a

[3] ICEMAN (Information-CEntric Mobile Ad-hoc Networking) architecture refers to a version of ENCODERS that does not include Adaptive Interest Modeling (AIM) or Rich Metadata Resolution (developed by Drexel University in the CBMEN program under another contract).

Random Waypoint Model. There is high network connectivity within a squad, but intra-squad connectivity is very limited. Occasionally squads pass one-another and induce brief periods of high connectivity.

Each node runs applications that generate different classes of traffic. We use three classes, defined in Table 1, for our evaluation, and each generates representative content to support situational awareness.

Table 1. Three classes of application-generated traffic, A1, A2, and A3, for use in evaluation.

	A1	A2	A3
Types	GPS; BioInfo1; BioInfo2	Area Pic.; Map; Audio Rec.	Frag Or.; Op Or.; Mission
Sizes (KB)	1;5;10	250;500;1000	251;501;1001
Pub. Dist.[4]	uniform., mean =10s	exponential., mean = 60s	exponential., mean = 300s
Total Pub.	1840	690	152
Max Recv.[5]	115200	2070	1520
Publishers	all	all	all
Subscribers	all	squad leaders	within squad

A1 traffic models a blue-force tracking application that generates small-sized content at a continuous rate (e.g., GPS coordinates or health vitals every few seconds). Everyone publishes content to everyone else. A2 traffic models a sensing application that collects data in response to random events, such as taking a photo of a suspicious vehicle, map annotations, or audio recordings. Content is pushed to the squad leaders who share the content with other squad leaders (content is pushed up the chain of command). A3 traffic models intra-squad communication (e.g., the squad leader pushes an operational order to the squad).

Each traffic class is subject to the following specified policies:

A1	A2	A3
RTTL (Relative Time-to-Live) of 10s	RTTL of 900s	RTTL of 400s
PO (Partial-Order) Replacement by creation time	No Replacement	No Replacement
Proactive Replication	Utility-based Replication	No Replication
No Network Coding	Network Coding	Network Coding

Phase 2 evaluation mainly focused on the new Phase 2 features, so we do not explore various transport policies nor Phase 1 dissemination policies. We use UDP broadcast without control as the transport policy. Data objects will only be delivered to nodes that are interested in them for A2 and A3 (i.e., no multi-hop routing through a relay). This selection is based on the Phase 1 evaluation [17]. We chose transport/dissemination policies that show the best performance according to our Phase 1 evaluation, augment them with new Phase 2 features, and evaluate our performance with the new features.

We evaluate ENCODERS for a variety of dissemination policies, which are described in Table 2. Each is a different composition of Phase 2 features.

[4] Inter-publication delay distribution.
[5] Maximum number of data objects that can be received given a fully connected network without mobility and resource constraints.

Table 2. The composite dissemination/caching policies that we will explore are defined in terms of the Phase 2 features that are added to the Phase 1 baseline policy.

Composite Policy	Composition of Features
Phase 1	Phase 1 Baseline
SP	Send Priority
SP+UR1	Send Priority, Utility-based Replication 1
SP+UR2	Send Priority, Utility-based Replication 2
SP+UC+UR1	Send Priority, Utility-based Caching, Utility-based Replication 1
SP+UC+UR2	Send Priority, Utility-based Caching, Utility-based Replication 2
SP+SC+UR1	Send Priority, Social-aware Caching, Utility-based Replication 1
SP+SC+UR2	Send Priority, Social-aware Caching, Utility-based Replication 2

For security, we evaluate two security policies; (i) dynamic provisioning and (ii) static provisioning. Dynamic provisioning is when nodes receive their certificates and encryption attributes at run time through the use of Security Data Requests and Responses over the network, see Section 4.5. This allows for configuration to be done at run time based on the scenario's requirements. When the network topology and security policies are known beforehand, however, it is more efficient if nodes do not have to wait for this request/response mechanism to occur at run time. In such a case, nodes can have the necessary certificates and encryption attributes configured beforehand; we call this process static provisioning.

Each content type is further subject to the specified policies. Table 3 shows the priority and utility assignment for each content type.

Table 3. The priorities and utilities assigned to the various types of content. The rightmost five columns are the dissemination policies that we evaluated.

Traffic Class	Content Type	Send Priority	Utility-based Replication 1	Utility-based Replication 2	Utility-based Caching	Social-aware Caching
A1	GPS	Very Low	No	No	Low	Low
	BioInfo1	Low	No	No	Low	Low
	BioInfo2	Low	No	No	Low	Low
A2	Area Pic.	Medium Low	High	Low	Medium	Medium
	Map	Medium	Medium	Medium	Medium	Medium
	Audio Rec.	Medium High	Low	High	Medium	Medium
A3	Frag Or.	High	No	No	High	High
	Op Or.	Very High	No	No	High	High
	Mission	Very Very High	No	No	High	High

The above scenario was tested on both Linux containers and Android devices (Nexus S). For Linux, we set a CPU limit for each virtual host to roughly match the CPU resources on the target phones. We show the parameter-space exploration results from varying policies. The metrics across all our experiments are total data objects delivered for each traffic class, data object delivery latency distribution for each traffic class, total transmit/receive bytes, and total data object delivered bytes (as described below).

Histogram Legend. Here we explain the terminology used in the graphs of this section. The x-axis of each histogram depicts results for different dissemination/caching policies

(see Table 2). Below the x-axis, the 5 types of bars appear in a legend: Tx, Rx, DRx, %A2-A3, and %A1, which we define here. For the first three, the y-axis (left side) is total bandwidth in MB. For the latter two, the y-axis on the right side gives the delivery fraction. The fraction of total data objects delivered for A2 and A3 is depicted by the cyan bar in histograms, and the fraction for A1 is depicted in the rightmost bar of each group of 5 bars. Delivery fraction is the ratio of delivered data objects over the theoretical maximum number of data objects that can be received given a fully connected network without mobility and resource constraints.

The bandwidth data are as follows: *total transmitted bytes (Tx)* and *total received bytes (Rx)* includes all traffic that occurs (i.e., both content and overhead used by ENCODERS to manage the network and deliver content), while non-track data-object received bytes (*DRx)* measures the *portion of Rx that constitutes* data objects in traffic classes A2 and A3 (i.e., the larger, non-track, higher-priority content) that are delivered to an application that is interested in them. Thus, comparing DRx to Rx shows the fraction of received bytes that are non-track content.

The results that can be expected for DRx and the delivery fractions depend on the scenario. Therefore, we include a baseline as the leftmost set of bars in each histogram. This leftmost group of bars is the results for ENCODERS at the end of Phase 1 for the given scenario. As described in Section 3, SRI's Phase 1 system had 2-2.5 times higher availability than the baseline centralized system (defined with the Government Test and Evaluation Team), with average latency about one-tenth of the baseline. Because our Phase 2 results improve on our Phase 1 results, the results shown in our histograms exceed current centralized networks by an even larger margin.

Latency-graph Legend. Data-object delivery latency is the amount of time that it takes the data object to be delivered to an interested subscriber after the data object is published. The x-axis is latency in seconds, and the y-axis is the total number of data objects that were received within that given latency. Each colored line represents the different compositions of policies, which focus to differing degrees on delivering the highest priority/utility content.

The latency graphs are comprised of four graphs in a quadrant. The top two, (a) and (b), are for data classes A2 and A3, and the bottom row, (c) and (d), is for all 3 data classes. The two in the left column, (a) and (c), are for the latency from subscription (i.e., interests injected into ENCODERS), and the right column is for the latency from publication (i.e., data objects injected into ENCODERS), see Table 4.

(a) A2+A3; Subscription to Delivery	(b) A2+A3; Publication to Delivery
(c) A1+A2+A3; Subscription to Delivery	(d) A1+A2+A3; Publication to Delivery

Table 4. Quadrants in a latency graph.

Linux Containers. Figure 17 through Figure 20 present the evaluation results on Linux containers using a network with 30 nodes.

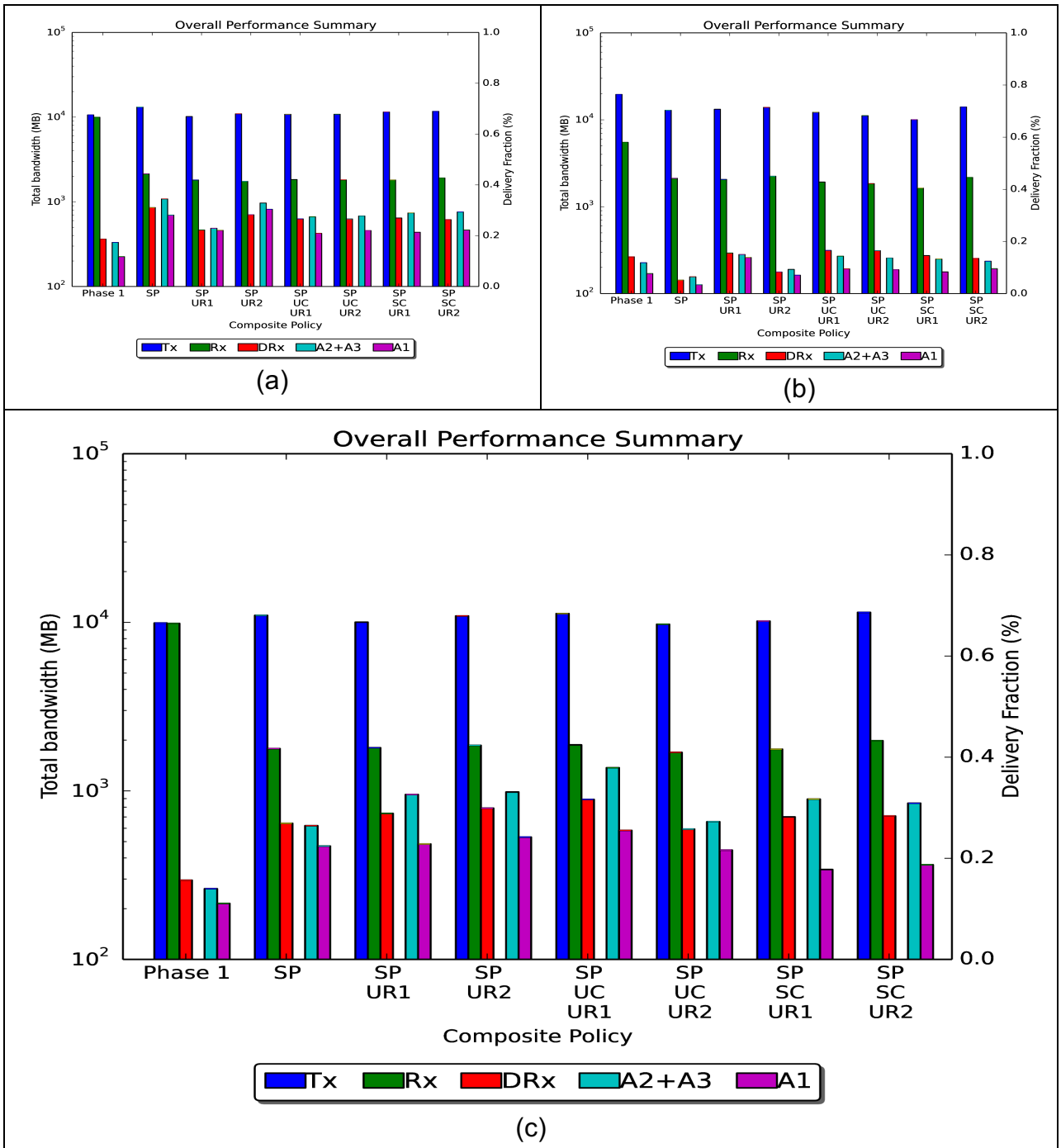


Figure 17. Data object delivery. The x-axis depicts results for different dissemination/caching policies (see Table 2), with 5 bars shown for each policy. The 3 graphs show the results for different security options: (a) Static Provisioning, (b) Dynamic Provisioning, and (c) No Security Policy. The 2 rightmost bars in each group show improved performance over our Phase 1 baseline (leftmost bars). The subfigure (b) shows a significant cost to using dynamic provisioning, while (a) shows almost no cost to using static provisioning.

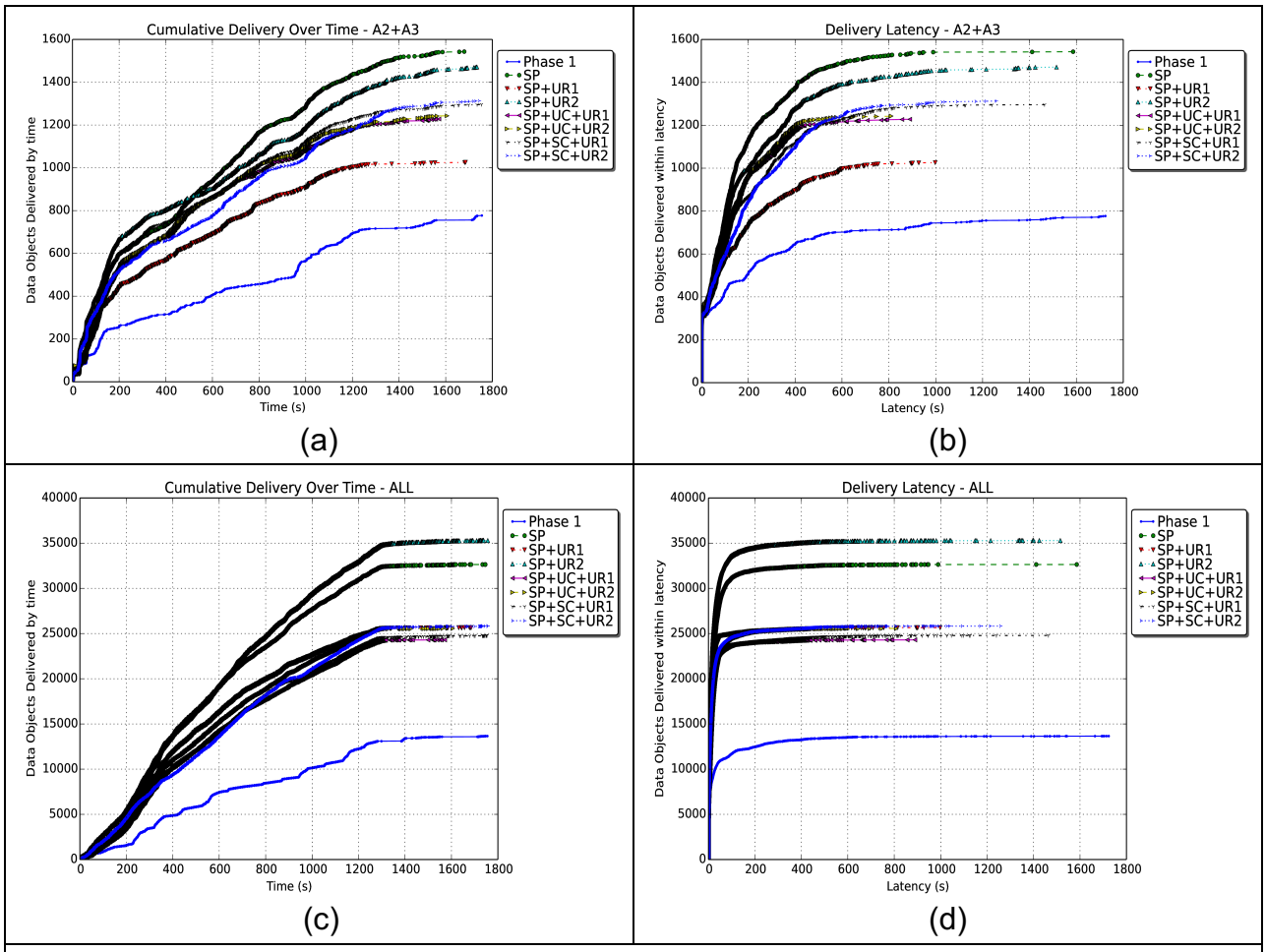
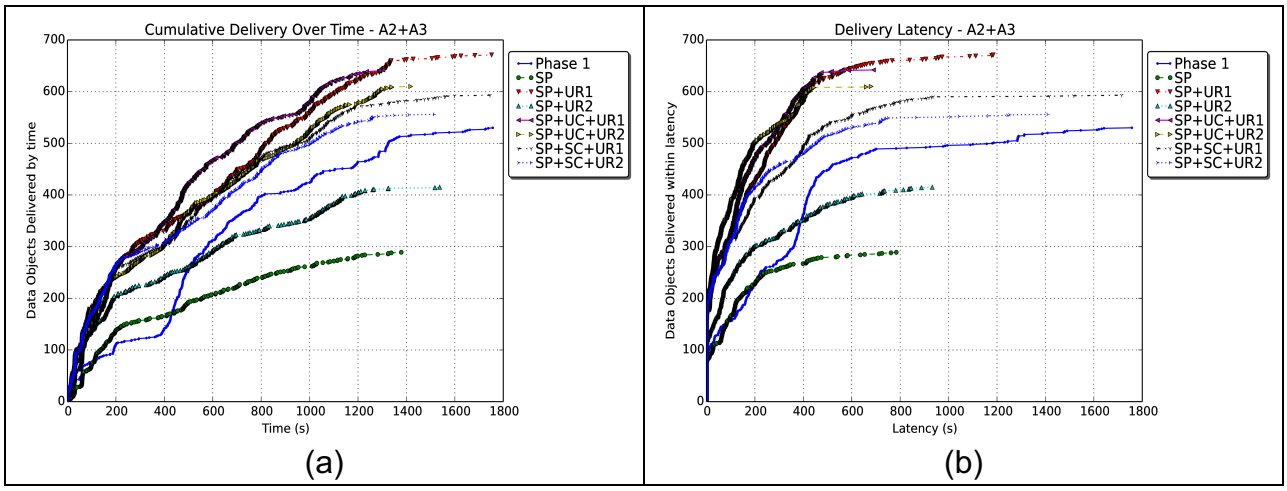


Figure 18. Delivery performance: (a) and (c) the cumulative number of data objects delivered (over the entire scenario), (b) and (d) within a given latency (in seconds on the x-axis). These results are for using the security option of *static provisioning,* on Linux containers. Each line represents a different policy combination.



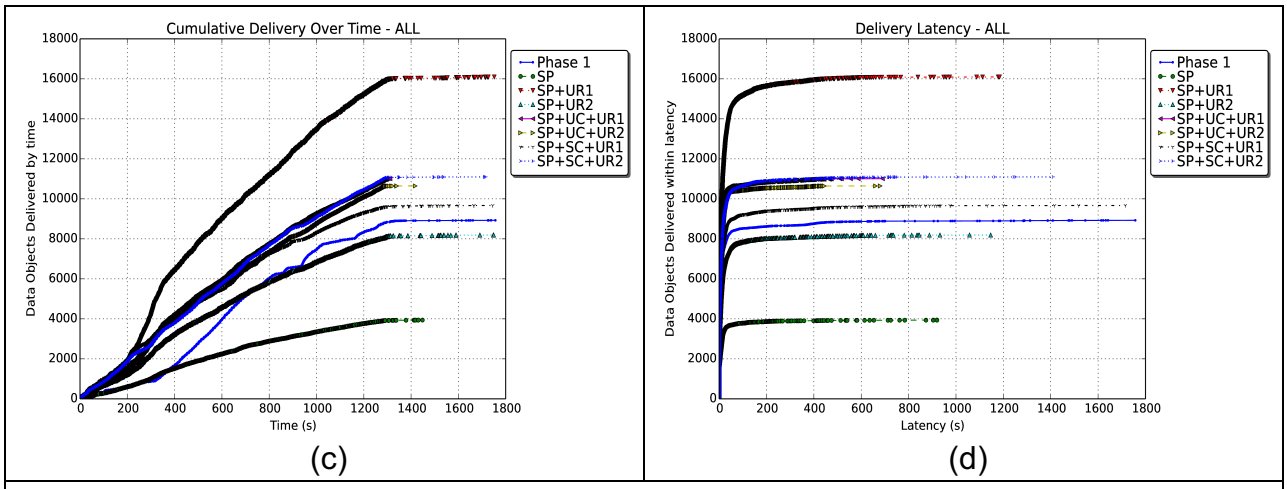


Figure 19. Delivery performance: (a) and (c) the cumulative number of data objects delivered (over the entire scenario), (b) and (d) within a given latency (in seconds on the x-axis). These results are for using the security option of *dynamic provisioning,* on Linux containers. Each line represents a different policy combination.

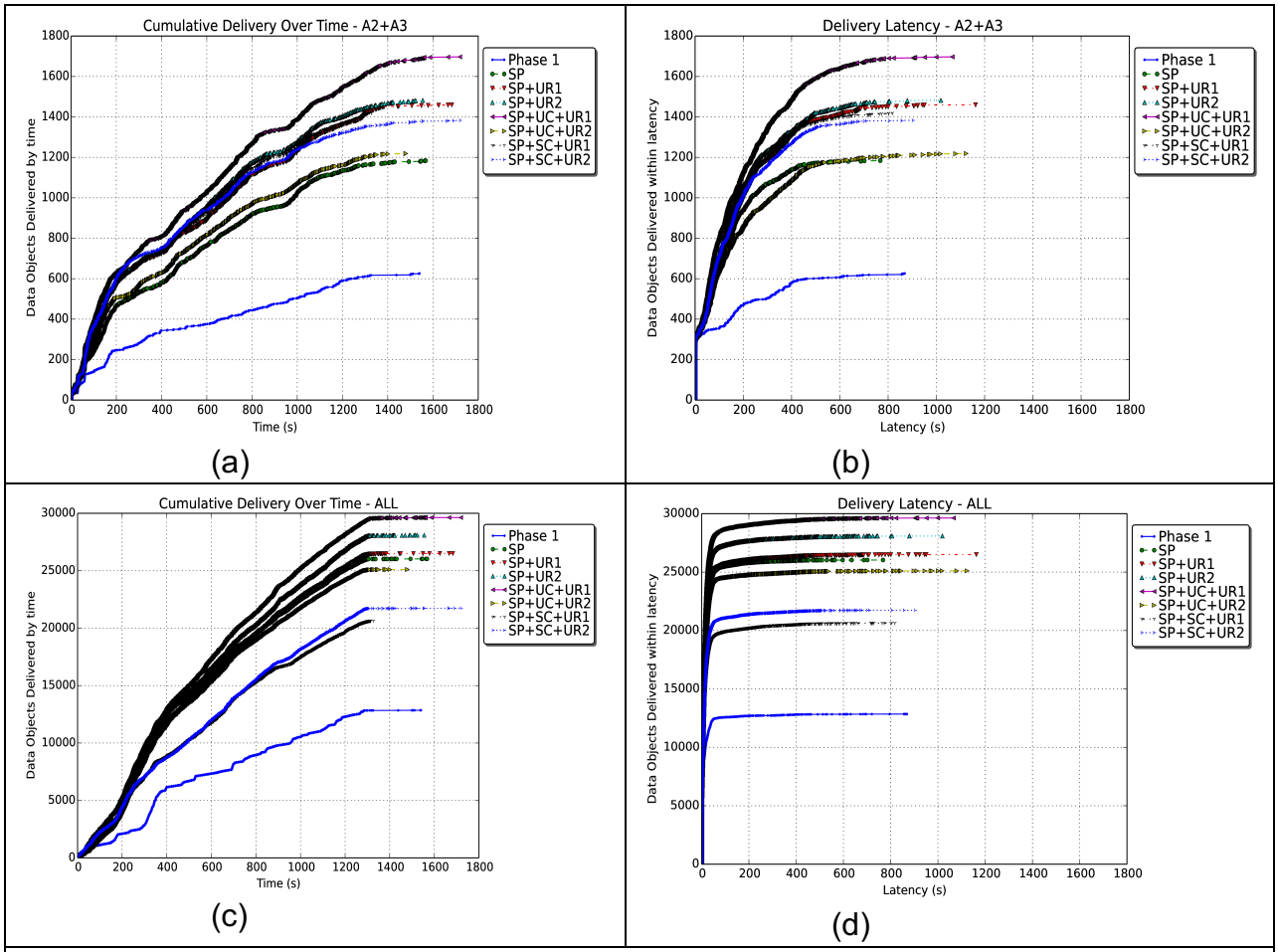


Figure 20. Delivery performance: (a) and (c) the cumulative number of data objects delivered (entire scenario), (b) and (d) within a given latency (in seconds on the x-axis). These results are for using no *security policy,* on Linux containers. Each line represents a different policy.

Android devices. Figure 21 presents the evaluation results on Android devices, first using a network with 30 nodes. We compare this figure to the Linux container results in Figure 17. In Phase 1, we see the A1 data delivery (purple bar) is much higher than it was for Linux containers, while the DRx and A2+A3 values are much lower. This is because network coding often blocks due to insufficient CPU on Android devices, resulting in all resources being devoted to delivering the small A1 data objects.

In Phase 2, our utility-based dissemination prioritizes A2 and A3 content, so most of the resources are spent on A2 and A3, with greatly improved throughput within the Android CPU limitations, as shown in the red and aqua bars of Figure 21. The A1 rates (purple bar) are low because outdated A1 data is not disseminated.

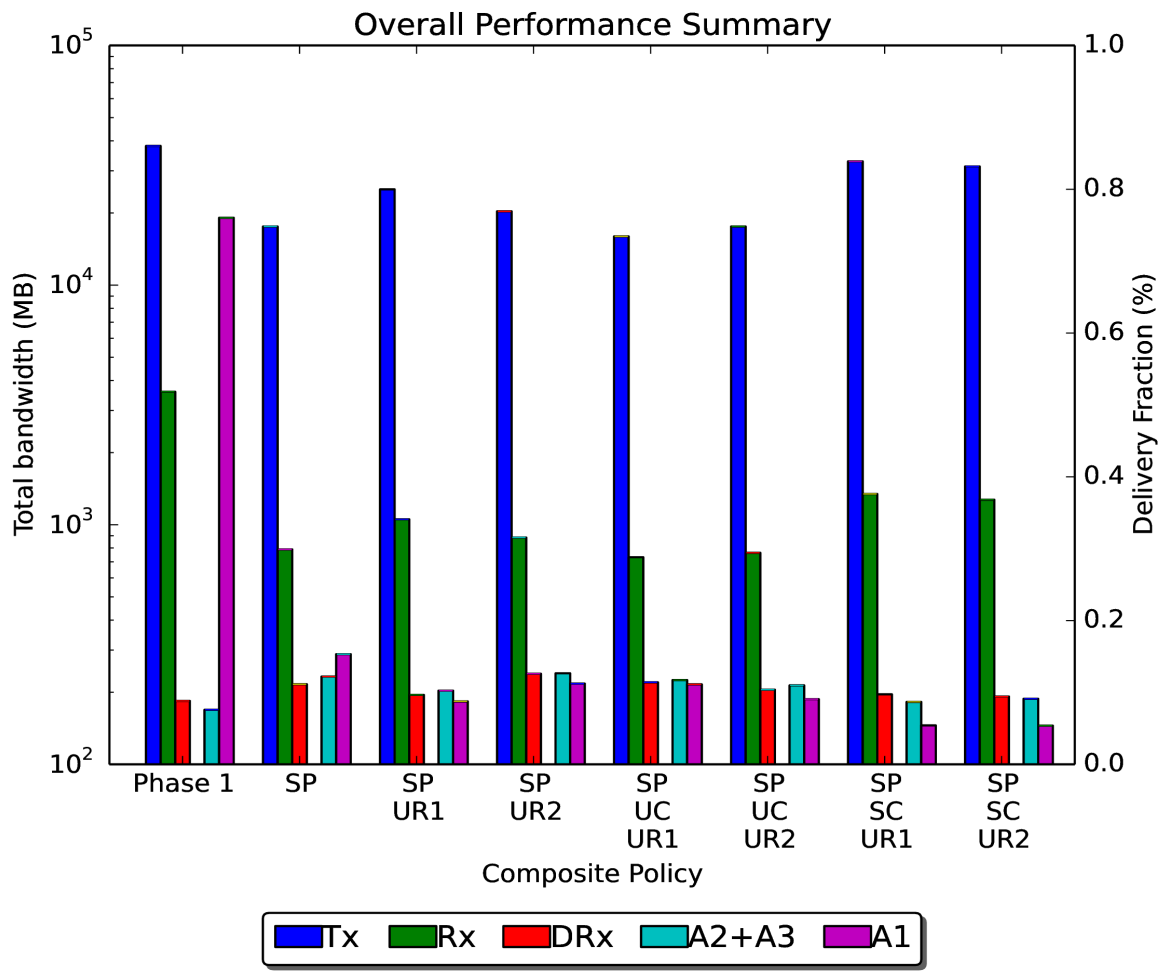


Figure 21. Data object delivery for different policies with static provisioning on Android devices in a 30-node network. Compared to the Phase 1 baseline (on the left), the 2 leftmost bars in each group indicate lower total usage of bandwidth for overhead for all policies, and the red and aqua bars in each group show improved performance for the higher priority A2 and A3 content for all policies.

Figure 22 presents the delivery performance on Android devices, for comparison to the Linux container results in Figure 18. We see the same effects. The different scale of the y-axis in Figure 18 (a,c) shows that 2 to 3 times more A2+A3 objects are delivered with Linux containers. The latency is lower with Androids, see Figure 22 (b,d), because the majority of the data objects never get delivered. The blue Phase 1 plot in subfigures (c) and (d) is caused by the same phenomenon as the tall purple bar in Figure 21—insufficient CPU for network coding means nearly all delivered objects are A1 data.

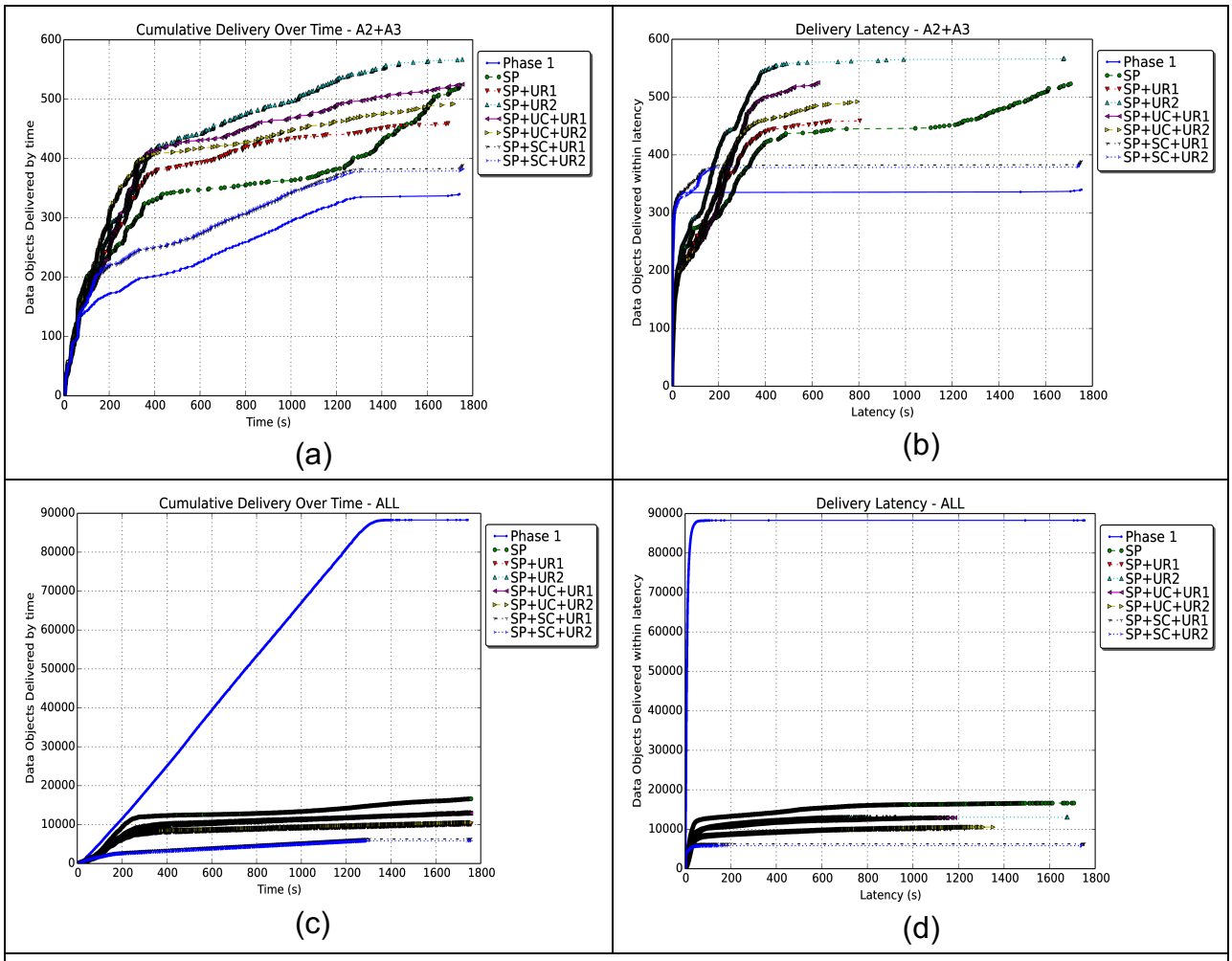


Figure 22. Delivery performance: (a) and (c) the cumulative number of data objects delivered (over the entire scenario), (b) and (d) within a given latency (in seconds on the x-axis). These results are for using the security option of *static provisioning,* on Android devices. Each line represents a different policy combination.

50-node Networks. We present the evaluation results on Linux containers with a network of 50 nodes in Figure 23 and Figure 24. Again, Phase 2 ENCODERS shows significant performance improvements over Phase 1 ENCODERS. Compared to the 30-node networks in Figure 17 and Figure 18, we see roughly the same total bandwidth, sometimes more, sometimes slightly less, depending on the policy (the y-axis is logarithmic). The delivery fraction is less because each data object has potentially 50 destinations instead of 30. The fact that ECODERS scaled from 30 to 50 nodes while maintaining throughput indicates that it scales better than many MANET approaches.

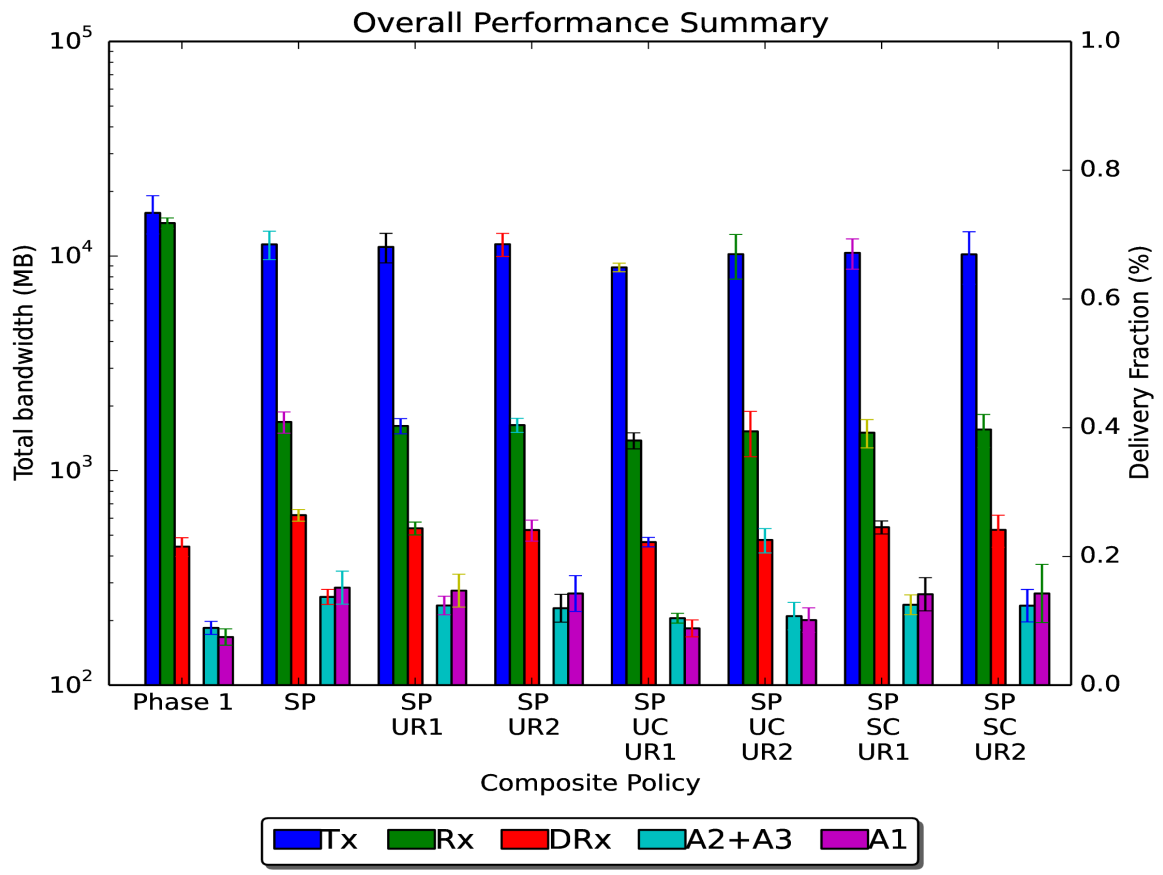


Figure 23. Data object delivery for different policies with static provisioning on Linux containers in 50-node networks. Compared to the Phase 1 baseline, the 2 leftmost bars in each group indicate lower total usage of bandwidth for overhead for all policies, and the 2 rightmost bars in each group show improved performance for all policies.

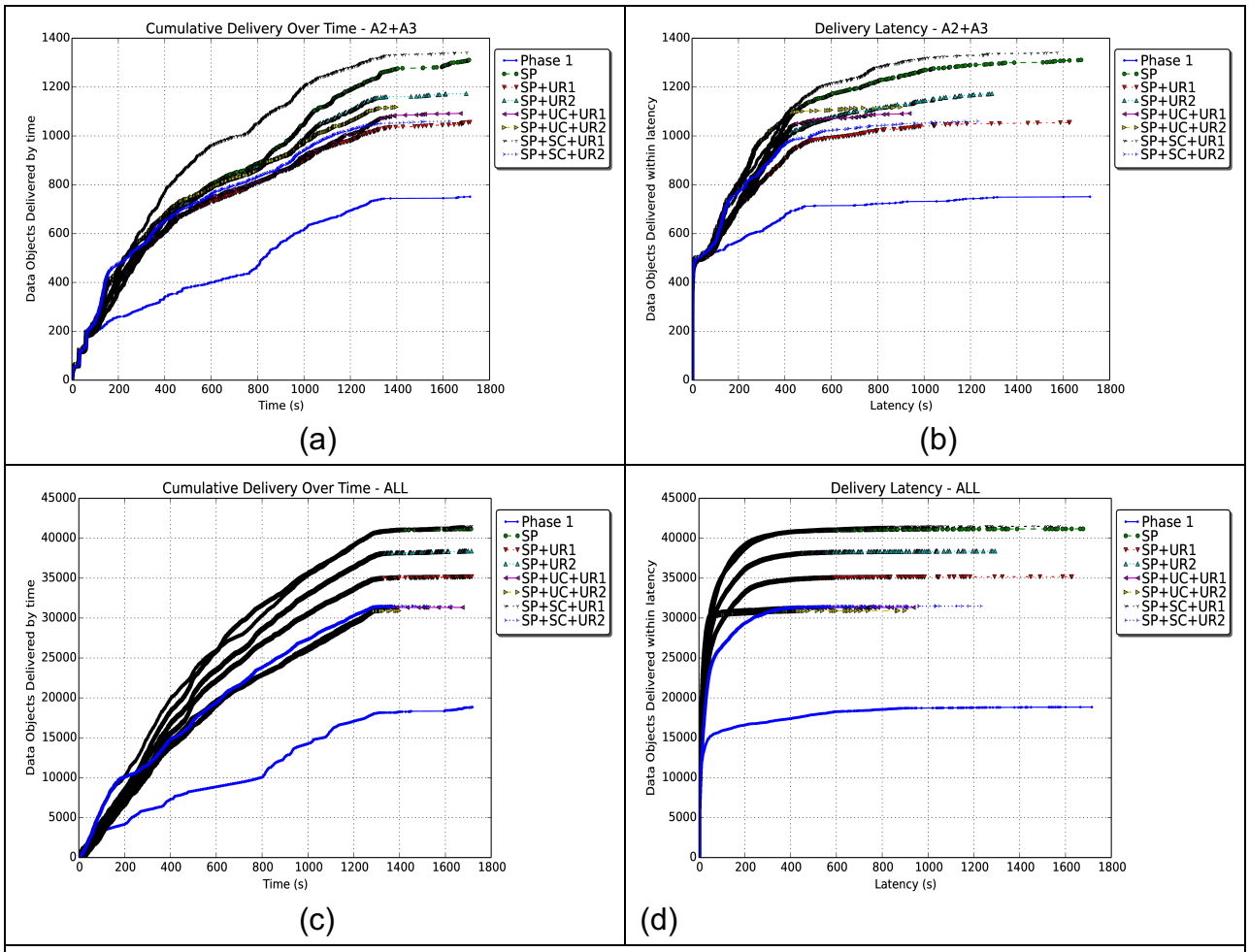


Figure 24. Delivery performance in a 50-node network: (a) and (c) the cumulative number of data objects delivered (over the entire scenario), (b) and (d) within a given latency (in seconds on the x-axis). These results are for using the security option of *static provisioning*. Each line represents a different policy combination.

Sensitivity to Cache Size. To understand what factors affect network performance, SRI conducted experiments to assess the sensitivity of our ENCODERS network to the cache size at each node. We will use Figure 17 (c) as the baseline, which depicts performance with an unrestricted cache. In this scenario, we observed that cache usage went up to approximately 32MB (in Phase 2) at many nodes, well within the cache limitations of proposed devices. Figure 25 shows the effect of limiting the cache to about ¾ of this typical cache usage. We see that the Phase 1 results are not affected. Because of the poorer performance in Phase 1, the cache rarely filled up, so limiting it had no effect.

Phase 2 ENCODERS delivers more data, using 32MB of cache at many nodes. With a 24 MB cache limit, we see delivery of A2+A3 reduced by 20-50%, although still significantly improved versus the Phase 1 baseline. This happens because data objects

are dropped before they can be delivered. These results indicate that ENCODERS performance is quite sensitive to cache size if the cache size is not sufficient for the content to be delivered in the scenario.

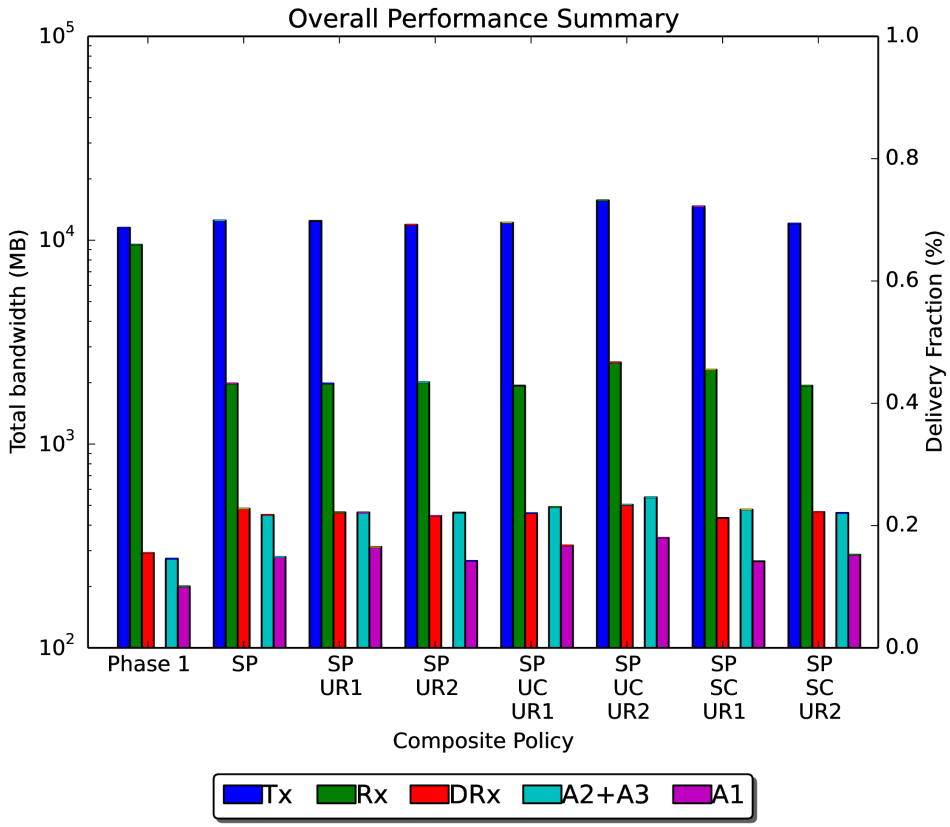


Figure 25. Data object delivery, 30-node Linux containers, with cache limited to 24MB. Compared to Figure 17 (c), we see delivery of A2+A3 reduced by 20-50%, although still significantly improved versus the Phase 1 baseline.

Figure 26 is another utility-based view of these results. In this analysis, we assign high/medium/low utilities to A3/A2/A1 content type, respectively. In general, such utilities might be assigned by the application producing them, taking into account the mission and the situation. Intuitively, a higher utility indicates more useful data was delivered. We see that the effects of limiting the cache size are significant, although performance is still better than Phase 1. The reduction in utility is most severe for the policies that perform best at delivering high-utility content.

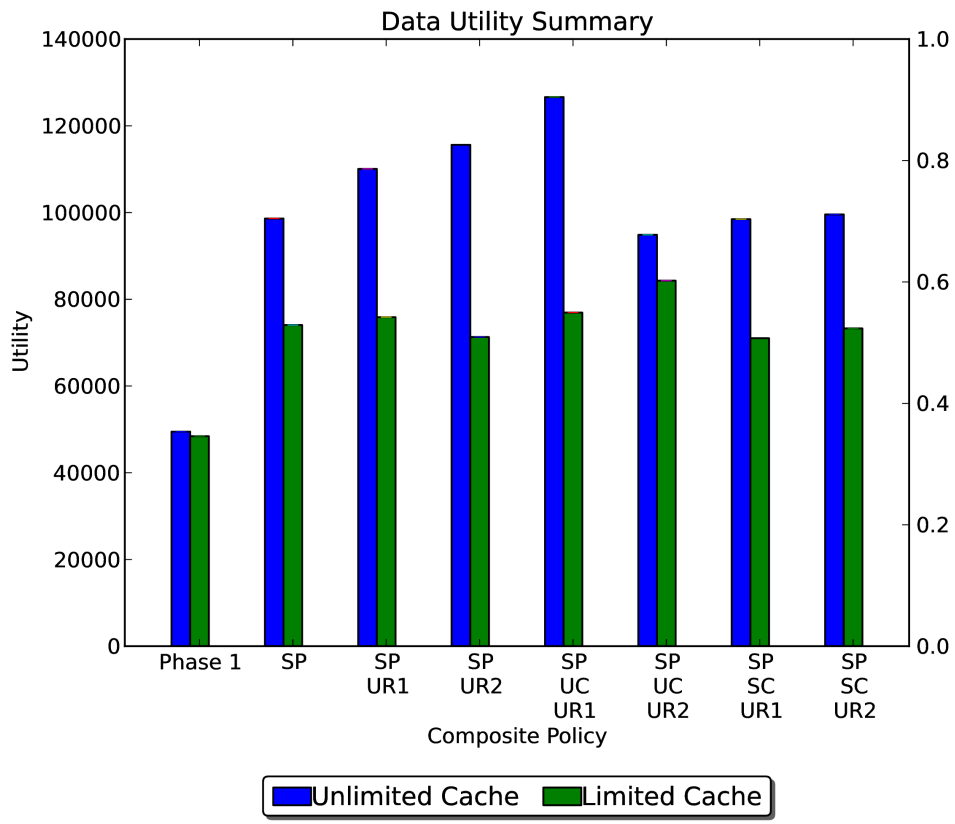


Figure 26. A utility-based view of the effects of limiting cache size. The effects of limiting the cache size are most severe for the policies that perform best.

5.3.2 Extended Scenario

Scenario. In Phase 2, SRI modeled a new military scenario for use in testing and evaluation to explore how widely applicable our system is beyond platoon-level ground operations. With guidance and direction from DARPA, we modeled a scenario that included ground, maritime, airborne, and satellite nodes.

Summary of Extended Scenario. Two 4-ships of fighters perform a reconnaissance mission. Each 4-ship is supported by a command and control (C2) aircraft in a racetrack orbit that serves as a relay to one ship and a ground site. The transmissions from fighters are limited, because they do not want to be detected by their transmissions and may be jammed, so the fighter pairs do not communicate with each other. The links from C2 aircraft to fighters and the link between one C2 aircraft and Ship have 50% connectivity (10 minutes on and 10 minutes off). We assume no connectivity from fighters to C2 aircraft.

Space-based nodes (such as nano-sats) support communication from fighters. The initial model contains two nano-satellites, which are always connected. Our initial model optimistically assumes a continuous satellite presence in the network. This corresponds to having enough satellites to provide continuous coverage and an ability to handoff network responsibilities from one satellite to another, which ENCODERS could support. In future work, one could model more realistic constellations of nano-sats with hand-over as constellations pass overhead, and parameterize links to explore different devices and scenarios.

We modeled the data exchange for a mission to collect radar image and video recording of the area of interest as follows (see
Table 5):

(1) Fighter-track: Each 4-ship shares air and ground Red/Blue/Gray tracks with the other 4-ship and the AWACS every 10 seconds. There are approximately 30-50 tracks shared; each track requires 150 bits. More recent track files supersede older track files.

(2) Radar Image: The two more forward fighters on the sensing mission generate a radar image, approximately 1Mbits, every 5 seconds, and share it with its associated AWACS, the surface ship, and ground site. Radar images are lower priority than tracks; more recent images supersede older images.

(3) Video Recording: Occasionally (every 5 to 10 minutes), the sensing fighter pair sends a compressed video stream to its associated AWACS, the surface ship, and ground site. We assume 64 kbps and the video recording lasts 1-2 minutes (7.8Mbits total). This is the lowest priority.

(4) Full Air Picture: Every 2 seconds, each AWACS broadcasts the full air picture, approximately 500 tracks (150 bits/track), to all nodes.

(5) Ground-track: The ground site sends 50 tracks every 2 seconds to all AWACS.

Table 5. Summary of the data exchange, followed by the policies specified for each type of content. (PO is partial-order replacement, RTTL is Relative Time-to-Live)

Content Type	Publisher	Subscriber	Size	Freq.
Fighter-track Full Air Pic. Ground-track	All Fighter Pairs All AWACS Ground	Fighter pairs in the other 4-ship, All AWACS Ship, The other AWACS, All Fighter Pairs All AWACS	7500bits 75000bits 7500bits	10s 2s 2s
Radar Image	Sensing Fighter Pair in each 4-ship	Associated AWACS, Ship, Ground	1Mbits	5s
Video Rec.	Sensing Fighter Pair in each 4-ship	Associated AWACS, Ship, Ground	7.8Mbits	10m

Content Type	Replacement	Purging	Send Priority	Network Coding
Tracks, Full Air Pic.	PO by creation time	RTTL of 60s	High	Off
Radar Image	PO by creation time	No Purging	Medium	On
Video Rec.	No Replacement	No Purging	Low	On

We modeled the Extended Scenario with node connectivity as shown in Figure 27.

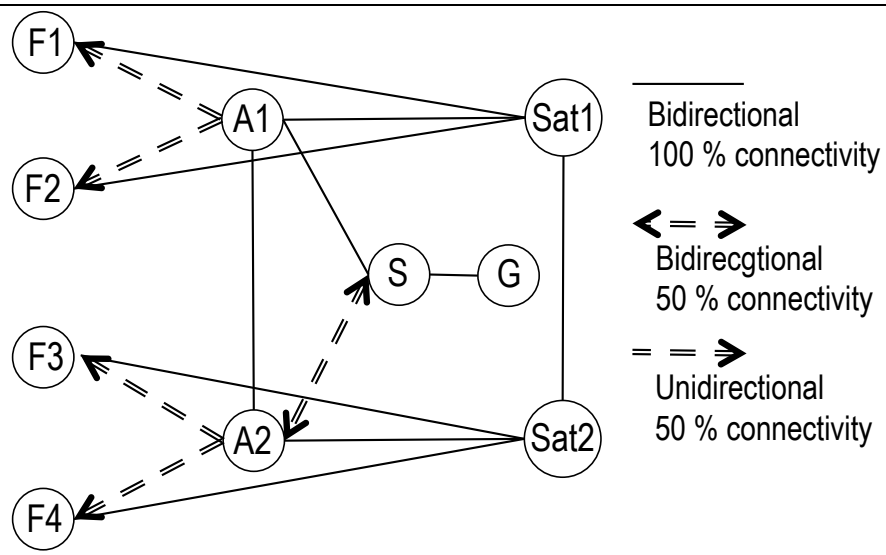


Figure 27. Airborne/ground network scenario — F1-F4: fighter pairs, A1-A2: C2 aircraft (e.g., E-3 AWACS), S: surface ship, G: ground site, Sat: satellite.

The link model is RF-Pipe with a data rate of 10Mbps. In future work, one could model more realistic constellations of nano-sats with hand-over as constellations pass overhead, and parameterize links (e.g., from Fighters to AWACS, between Ship and AWACS) to explore possible operating scenarios.

Results. The following figures present the initial performance results in the Extended Scenario, with every node in the network using ENCODERS, with its cache replacement option used for tracks and radar (i.e., outdated data objects can be replaced). We present results for runs with and without the satellite nodes.

Figure 28 (a) shows that, with satellites, 69% of tracks are delivered within 2 seconds and 100% within 12 seconds, while 66% of radar is delivered within 20 seconds. Videos take approximately ten minutes, a time that is likely high because of the transmission being interrupted while in progress. The graphs on the right side have a logarithmic y-axis, so the first bar represents over 10^4 tracks.

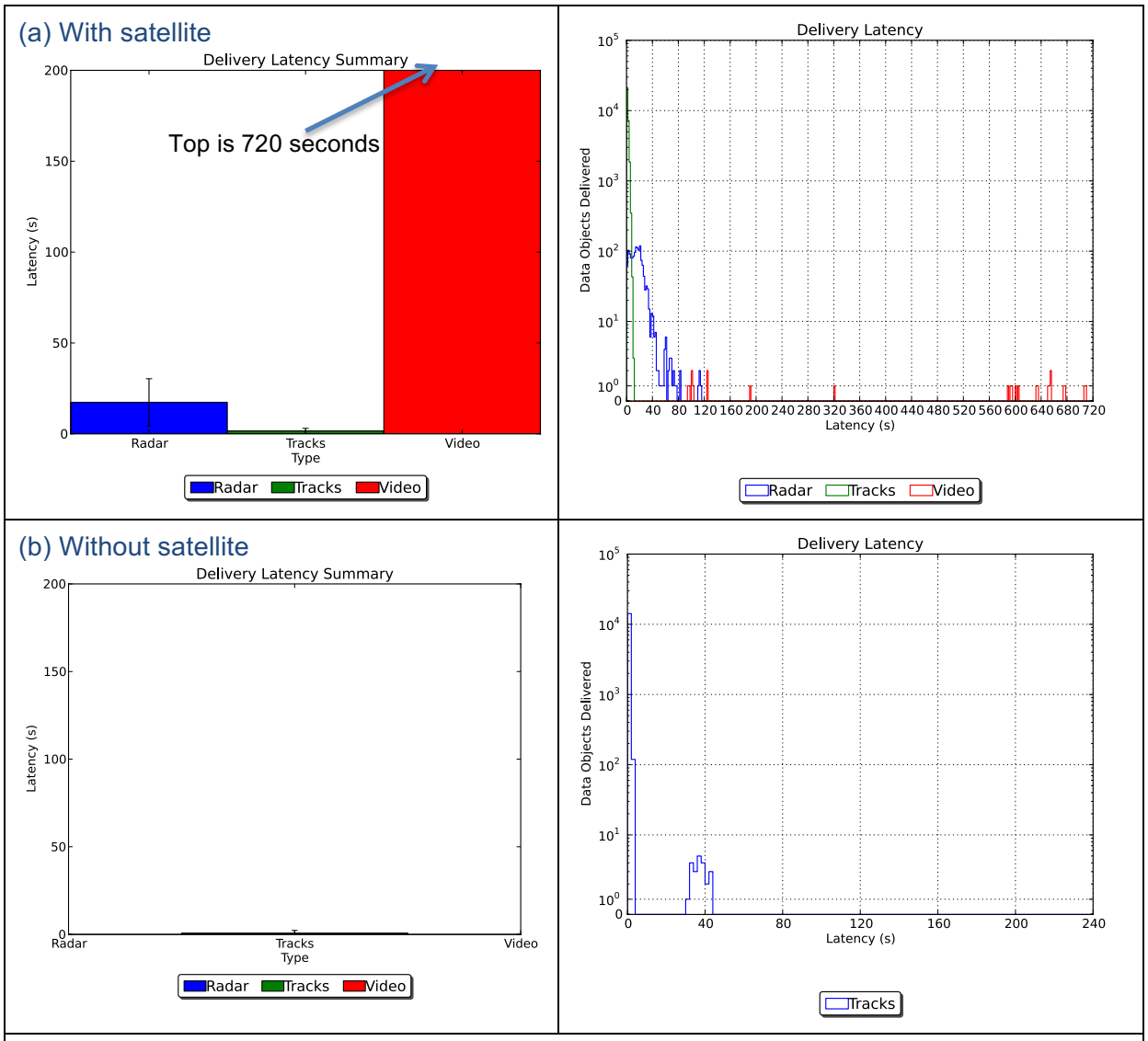


Figure 28. Data-object delivery latency for different types of content, (a) with the satellite, and (b) without the satellite. Without the satellite, only tracks get delivered.

Without satellites, Figure 28 (b) shows that only tracks get delivered, mostly within 2 seconds. Looking more closely at the tracks, only 14,356 tracks are delivered versus

30,756 tracks with the satellite. The lower performance without satellites is due to disconnections (as nodes move) and the use of the ENCODERS utility-based cache replacement, which will discard undelivered tracks when they are outdated. Radar and video are not delivered because fighters choose not to broadcast to non-space neighbors, to avoid detection.

Figure 29 shows data-object delivery for different types of content. In (a), videos are published every ten minutes by the fighters and received at the AWACS (blue line in the graph) at regular intervals. Videos arrive at the ground and ship (green and red lines) at approximately the same times because of the reliable link between the ship and ground. About half the time, there is a significant delay between the video getting to the AWACS and subsequently getting to the ship, which is due to periods of disconnection while the AWACS is out of range of the ship.

Radar data is received at regular rates at the AWACS as shown in Figure 29 (b). The maximum possible number of radar images received at the two AWACS combined is twice that at the ground or ship (as there is only one of each). The radar images received at ground and the ship are significantly less than this maximum, because outdated images get replaced when lack of connectivity prevents their delivery. Delivery at the ground lags that of the ship as delivery to ground goes through the ship.

The tracks are delivered at a regular rate shown in Figure 29 (c), because they are highest priority and travel along stable links. In (d), tracks from the ground to the AWACS go at a constant rate as one AWACS is consistently connected to the ship and it can forward them on to the other AWACS as well. Tracks from the AWACS to the ship and each other go at a constant rate due to the same reason. (Thus, lines for each of these overlay each other in the graph). Tracks from the AWACS to the fighters go at a constant rate while they are connected but there are 10 minute periods where no tracks go through due to disconnection, producing the discontinuity seen in the green-diamond lines in the graph. There is no burst when the AWACS come back in contact, because ENCODERS has replaced all outdated tracks with the latest information.

Conclusion. With satellites, the other nodes in the network gain, without the risk of the fighters communicating to the AWACS, knowledge of the fighters' location (mostly within 2 seconds), and of the content published by the fighters (2/3 of radar delivered within 20 seconds, videos take approximately ten minutes). Figure 30 shows the delivery rate of data objects over time with and without satellites. We see almost double the delivery, and consistent delivery, due to the satellites ensuring contact between nodes. ENCODERS is also able to deliver radar and video because of the satellite.

Without the satellite, tracks from AWACS to fighters can only be transmitted when they are in contact every ten minutes, producing the regular intervals of higher/lower delivery seen in Figure 30. There is a consistent delivery without satellites of tracks from ground to AWACS and from AWACS to the other AWACS and the ship, but no delivery of content published by the fighters.

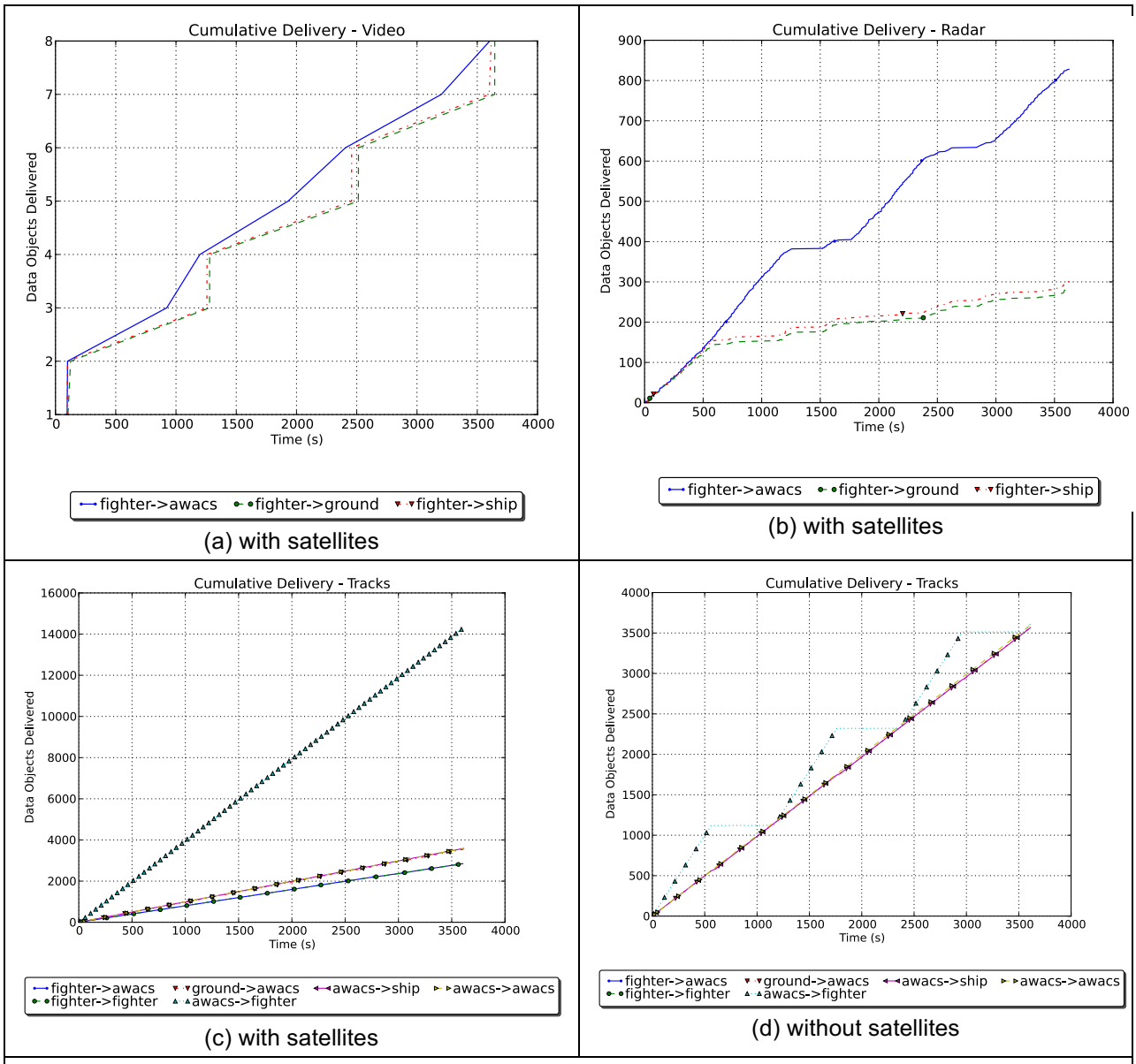
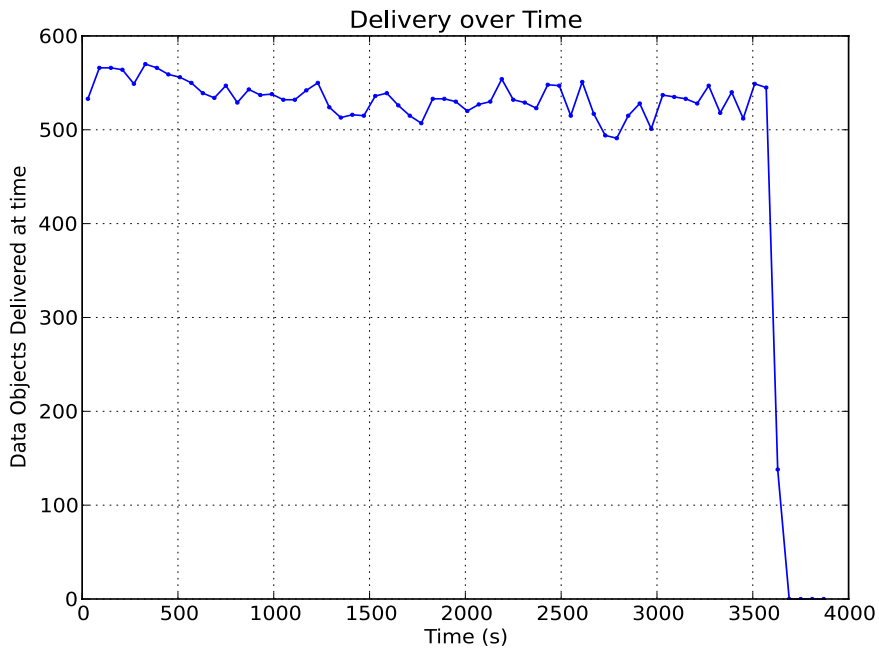
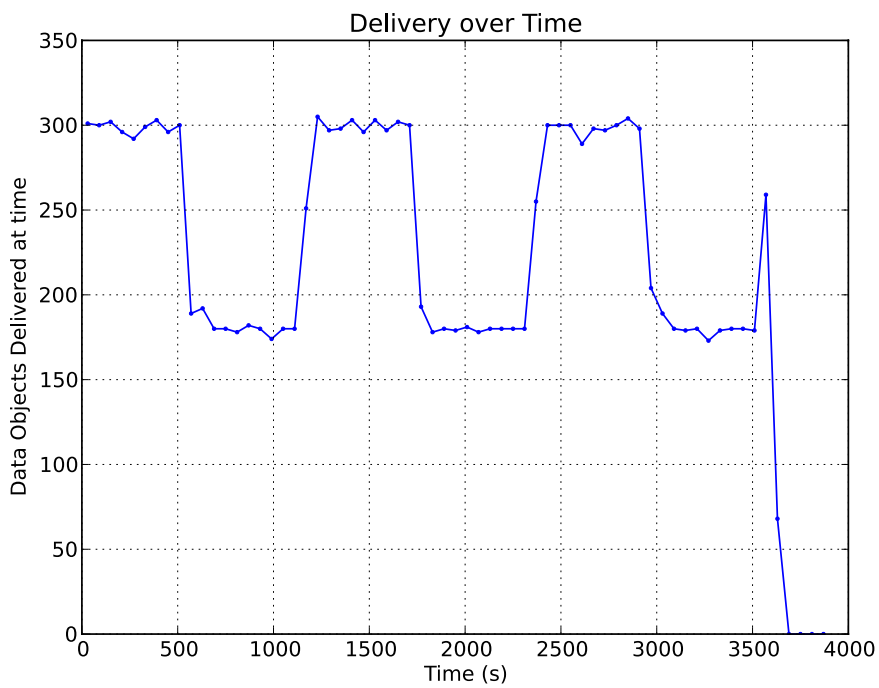


Figure 29. Data object delivery by type with satellites. This shows the cumulative number of data objects received over time between each pair of publisher/subscriber (represented by the different colors and icons in the legend). The subfigure (a) shows video, (b) shows radar, (c) shows tracks with the satellite, and (d) shows tracks without the satellite. Note the large differences in scale of the y-axis for each graph.



(a)



(b)

Figure 30. The number of data objects delivered in the previous minute at time t, (a) with the satellite, and (b) without the satellite. Note the different y-axis scales: (a) shows 500+ data objects being consistently delivered, while (b) alternates between about 175 and 300, as the AWACS connection comes and goes.

Bibliography

1. U. Lee, J. S. Park, J. Yeh, G. Pau, and M. Gerla. "Code torrent: content distribution using network coding in VANET." International workshop on Decentralized resource sharing in mobile computing and networking, ser. MobiShare ’06. 2006.
2. A.G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh. "A survey on network codes for distributed storage." Proceedings of the IEEE 99, no. 3 (2011): 476-489.
3. Ahlswede, R. Fakultat fur Math. "Network Information Flow." IEEE Trans. Info. Theory 46, no. 4 (2000).
4. Allison Lewko, Brent Waters. "Decentralizing attribute-based encruption." EUROCRYPT. 2011.
5. Anders Lindgren, Avri Doria, and Olov Schelen. "Probabilistic Routing in Intermittently Connected Networks." ACM SIGMOBILE Mobile Computing and Communications Review, 2003.
6. Erik Nordstrom, Per Gunningberg, and Christian Rohner. A Search-based Network Architecture for Mobile Devices. Uppsala University, Technical Report 2009-003, 2009, http://user.it.uu.se/~erikn/papers/HaggleArch.pdf.
7. Erik Nordstrom, Per Gunningberg, and Christian Rohner. "Haggle: Relevance-Aware Content Sharing for Mobile Devices Using Search." 2012, http://user.it.uu.se/~erikn/papers/haggle-arch.pdf.
8. Franca Delmastro, Silvia Giordano, Hoang Anh Nguyen, Erik Nordstrom, Melek Onen, Andrea Passarella, Alessandro Puiatti, Christian Rohner, George Theodorakopoulos, and Salvatore Vanini. "Specification of the CHILD-Haggle." 2007.
9. Franklin, Dan Boneh and Matt. " Identity-based encryption from the Weil pairing." SIAM Journal on Computing, 2003.
10. Ignacio Solis, J.J. Garcia-Luna-Aceves. "Robust content dissemination in disrupted environments." Proceeding CHANTS '08 Proceedings of the third ACM workshop on Challenged networks, 2008.
11. J.-S. Park, M. Gerla, D. S. Lun, Y. Yi, and M. Medard. "Codecast: a network-coding-based ad hoc multicast protocol." Wireless Communications. 2006.
12. John Bethencourt, Amit Sahai, and Brent Waters. "Ciphertext-policy attribute-based encryption." IEEE Symposium on Security and Privacy. 2007.
13. K. Obraczka T. Spyropoulos, T. Turletti. "Routing in delay-tolerant networks comprising heterogeneous node populations." IEEE Transactions on Mobile Computing, 2009.
14. Mark-Oliver Stehr, Carolyn Talcott. "Planning and learning algorithms for routing in disruption-tolerant networks." IEEE Military Communications Conference. 2008.
15. Minyoung Kim, Mark-Oliver Stehr, Jinwoo Kim, Soonhoi Ha. "An Application Framework for Loosely Coupled Networked Cyber-Physical Systems." 8th IEEE/IFIP Conference on Embedded and Ubiquitous Computing (EUC'10). 2010.
16. S.-H. Lee, U. Lee, K.-W. Lee, and M. Gerla. "Content distribution in VANETs using network coding: The effect of disk I/O and processing O/H." Sensor, Mesh and Ad Hoc Communications and Networks, 2008. SECON ’08. IEEE Communications Society Conference on. IEEE, 2008.

17. Sam Woods, James Mathewson, Josh Joy, Mark-Oliver Stehr, Minyoung Kim, Ashish Gehani, Mario Gerla, Hamid Sadjadpour, J.J. Garcia-Luna-Aceves. “ICEMAN: A System for Efficient, Robust and Secure Situational Awareness at the Network Edge,” in Proc. *32nd IEEE Military Communications Conference (MILCOM’13)*, Nov. 2013.
18. Hua Li, Ralph Costantini, David Anhalt, Rafael Alonso, Mark-Oliver Stehr, Carolyn Talcott, Minyoung Kim, Timothy McCarthy, and Sam Wood. "Adaptive Interest Modeling Enables Proactive Content Services at the Network Edge", UMAP ProS 2014 workshop, July, 2014.
19. Sam Woods, James Mathewson, Josh Joy, Mark-Oliver Stehr, Minyoung Kim, Ashish Gehani, Mario Gerla, Hamid Sadjadpour, J.J. Garcia-Luna-Aceves. "ICEMAN: A Practical Architecture for Situational Awareness at the Network Edge", Available online at http://encoders.csl.sri.com.