CBMEN ENCODERS

Content-Based Mobile Edge Networking Program

Edge Networking with Content-Oriented Declarative Enhanced Routing and Storage

Contract Number: N66001-12-C-4051

ENCODERS
Software Design Description
Version 2.0 – *September 2, 2014*

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. Carolyn Talcott, Phone: (650) 859-3044, Email: clt@csl.sri.com

Table of Contents

1 Scope

The purpose of this document is to describe the CBMEN ENCODERS (Edge Networking with Content-Oriented Declarative Enhanced Routing and Storage) software design. Section 4 presents the concept of operations (CONOPS) for using our design for security in the domain of small-unit operations of warfighters at the edge.

This document first 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. Next, there is a section that describes the design of each major enhancement that SRI made, and, when appropriate, these sections discuss other alternative solutions and their pros and cons, as well as suggested future improvements.

Evaluation results for ENCODERS can be found in SRI's CBMEN Final Report, and detailed documentation can be found in (1) the ENCODERS Configuration Manual, which is intended for system administrators who are setting up an ENCODERS network, (2) the ENCODERS Technical Documentation, which is intended for developers who may be enhancing or modifying the system (e.g., describes software classes and events), and (3) the Haggle documentation at https://code.google.com/p/haggle/ .

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 Labs 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.

1.3 SRI's Technology Development

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 3.

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

ENCODERS has also leveraged results from SRI's current NSF- and ONR-funded projects on Networked Cyber-Physical Systems, and ideas developed by International

Fellows in their own research while at SRI, including Hasnain Lakhani, Jong-Seok Choi, Dawood Tariq, Rizwan Asghar, Je-Min Kim, and Francoise Sailhan.

1.4 Value Proposition and Measures

Early on, DARPA requested that we develop a Value Proposition for CBMEN 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 primary system performance measures against this value proposition are:

- **Accuracy:**
    - The percentage of "relevant information" that is delivered.
    - The percentage of "non-relevant" information that is delivered.

  These metrics are intended to measure the capability of CBMEN to accurately represent the users' interest and use such representation effectively to deliver the right content.

- **Latency:**
    - For one-shot queries, the time from a query (given the information is in the system) to the time the relevant response is received.
    - For standing subscriptions, the time from when the relevant information becomes available (assuming the interest is registered) until it is delivered to the interested party.

Although accuracy and latency are the highest-level system performance metrics, they are not the only ones. The secondary metrics are measures such as bandwidth utilization, power consumption, computational power utilization, and memory utilization. These are all essentially measures of system efficiency. These measures are important and are discussed in our Final Report.

2 Overview of ENCODERS Design within CBMEN

2.1 Overall Architecture

The objective of the CBMEN program was development of 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 SRI's CBMEN content-based networking solution. 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.

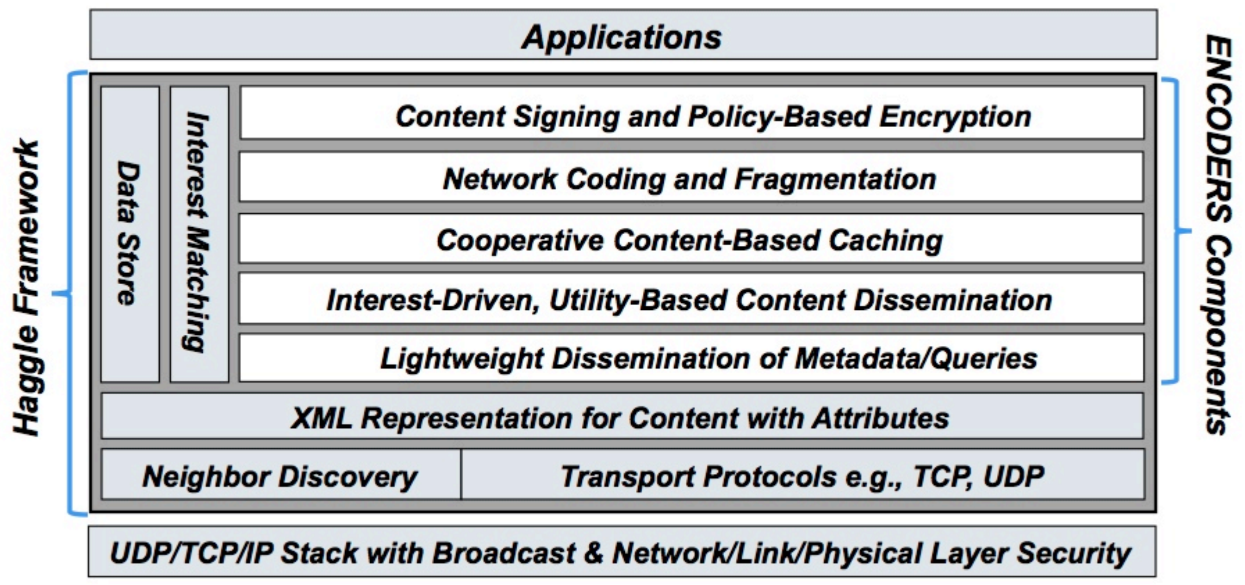
Applications
Haggle Framework
Data Store
Interest Matching
Content Signing and Policy-Based Encryption
Network Coding and Fragmentation
Cooperative Content-Based Caching
Interest-Driven, Utility-Based Content Dissemination
Lightweight Dissemination of Metadata/Queries
XML Representation for Content with Attributes
Neighbor Discovery
Transport Protocols e.g., TCP, UDP
UDP/TCP/IP Stack with Broadcast & Network/Link/Physical Layer Security
ENCODERS Components

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 TCP/IP to connect to peers. Section 2.2 gives a brief overview of the ENCODERS architecture and its leveraging of Haggle, and Section 0 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, Protocols, 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.

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.

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

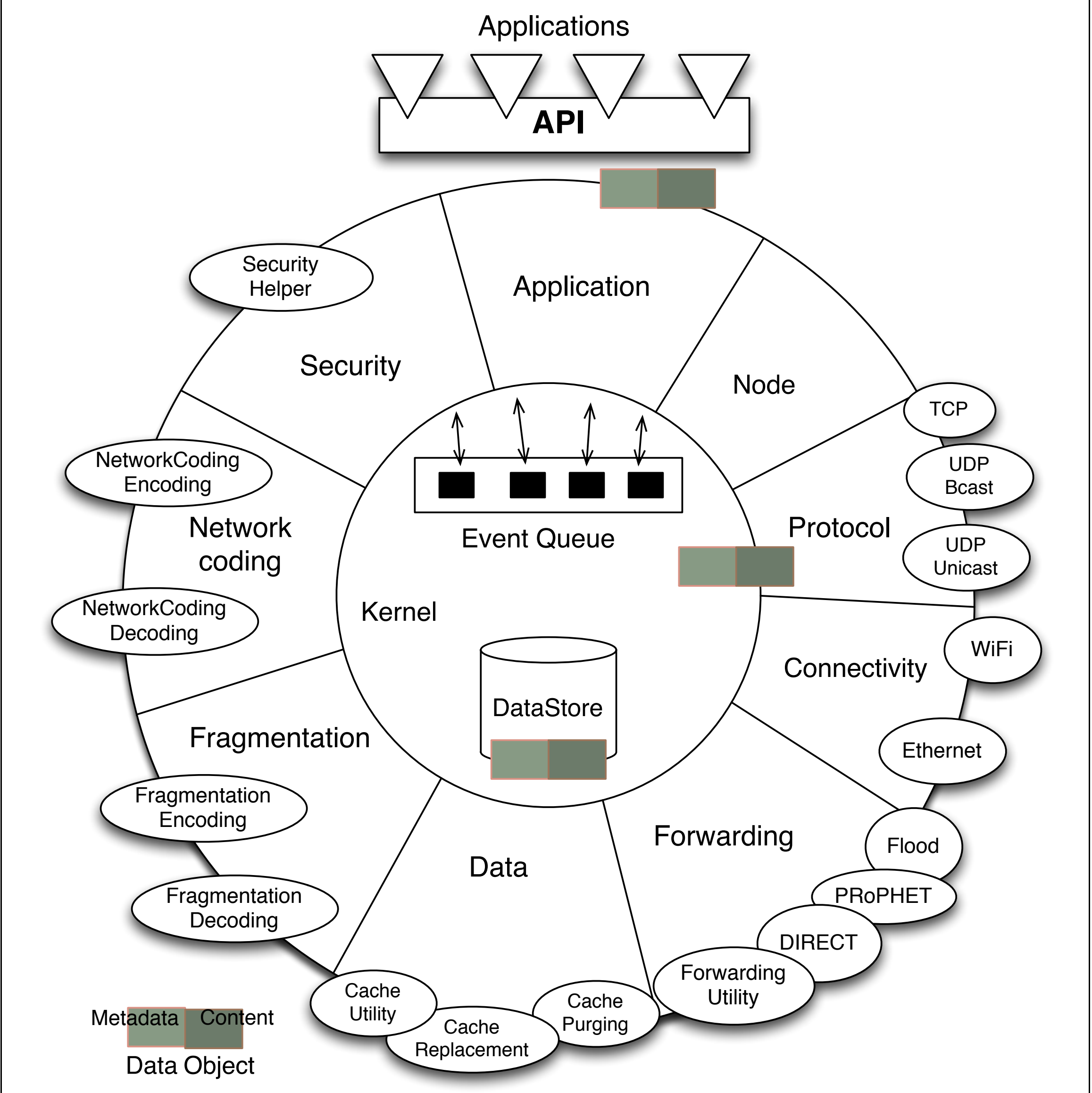

Figure 2. ENCODERS Architecture and API. The circle depicts the layer-less interaction. Each wedge of the circle is a *Manager*, and they interact through the Event Queue and DataStore in the Kernel (center). The managers may use the options provided by the modules in the small circles. Not all managers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

Upon receiving a new node description or 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.

Data objects can be inserted in the data store either by a local application, or by receiving a data object from a neighbor. 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 ENCODERS Software Design Description

The technology development performed by the SRI team can be grouped into areas that extend the Haggle framework in orthogonal dimensions. A separate section in this document 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 (Ignacio Solis 2008), which was developed in the DARPA Disruption-Tolerant Networking (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 and context-awareness.

- **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 *not* on the attributes (so that the attributes can still be matched with interests) or 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 describes the concept of operations (CONOPS) for small unit operations for the ENCODERS security solution.

3.1 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 3.2.

3.1.1 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.

DIRECT uses flooding to propagate interests, as describe in our Technical Documentation. Nodes discover each other using periodic `HELLO` messages to maintain a neighbor table. If a node hears a 1-hop neighbor, then it marks it as active. If a node no longer hears a neighbor after a period of time, then it marks the neighbor as remote. When a node is marked active, the two nodes exchange query information to propagate queries that have not been answered yet.

Design and Implementation. We designed our DIRECT module to leverage the functionality already provided by Haggle, especially the fact that each node maintains knowledge about other nodes and the content they have already received and cached. Additionally, our approach is extensible and was easily implemented, so that we could rapidly prototype different variations of DIRECT and quickly uncover unforeseen problems. To support a large class of scenarios and improve performance, we implemented an extensible mechanism of distributing content proactively, without explicit user subscriptions.

To evaluate DIRECT's effectiveness, we implemented multiple versions: Alpha-DIRECT and Alpha-DIRECT++. All of these versions support all-match semantics, and they are implemented as delegate forwarders, with slight modifications to the Forwarding Manager. DIRECT has two phases: 1) the interest flood, and 2) reverse path data forwarding. Haggle already provides a mechanism for distributing node descriptions that contain an interest list, based on interest commonality. Below, we describe our design for implementing DIRECT by leveraging Haggle.

For Alpha-DIRECT, we exploited Haggle's one-hop propagation mechanism, by making sure all nodes share a common interest attribute, such as key = "NodeDescription", value = "", weight = 1[2]. When a node's interest changes, it will trigger a flood of its node description (and consequently its interests) to all nodes within a connected component (as DIRECT does when a node floods an interest). In this version, when a node's interest list changes or a new data object is added its node description it is re-flooded (in the same way DIRECT sends a flood when it has a new interest, but without propagation across connected components). Haggle's use of Bloom filters prevents an already received node description from being retransmitted.

To address reverse-path data forwarding, Haggle remembers from which neighbor it received a node description (and consequently, the node's interest). Whenever Haggle receives a data object, it records the remote interface from which it received the object. Thus, we do not need to add another data structure (i.e., an interest direction table) to keep track of the source of the interest; we can inspect the existing node store to find the delegate (one-hop neighbor) node that belongs to the remote interface for the target. Similarly, when generating a list of targets for a particular delegate, we can use the data store and node store to find all the node descriptions that were received through that neighbor. By not introducing a new data structure, our design reuses the

[2] This design allows us to reuse an abstract version of Haggle's native node description attribute (which in unmodified Haggle has the node identifier as its value.

aging and cleaning mechanisms already in place for the data store. The Technical Documentation has implementation details.

Limitations of Alpha-DIRECT. Alpha-DIRECT is a bare-bones prototype that has limitations both in its implementation and design. It requires the use of a shared attribute for all nodes in the network for node descriptions to propagate (as they contain a node's interest and Bloom filter). This trick makes testing more difficult, as an application must register a hard-coded interest. More generally, the shared attribute has unclear semantics when other applications also have an interest in the attribute. Similarly, if a node has set a maximum number of matches, then the hard-coded attribute can potentially change the existing Haggle semantics: some files may or may not be transferred when DIRECT is enabled or disabled.

With respect to the design, Alpha-DIRECT did not refresh or explicitly timeout existing node descriptions. Traditional DIRECT uses a soft-state approach of re-flooding interests in order to find new paths and maintain existing paths. This re-flooding is based on a timer (20 seconds in the original paper). In contrast, Alpha-DIRECT's use of a shared attribute will only flood a node's description when its interest changes. Thus, our initial version performed poorly when the network topology changed, because alternative paths were not discovered automatically.

In response to these limitations, we developed Alpha-DIRECT++. 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).

Limitations of Alpha-DIRECT++. Although Alpha-DIRECT++ can support a variety of scenarios, it still relies on the publisher to receive an explicit interest from the sender before disseminating content. To support a larger class of scenarios, we extended the flooding mechanism primarily used for node-description dissemination to support a basic form of proactive replication.

Another limitation was using the data object abstraction as the fundamental unit of content dissemination. Alpha-DIRECT++ would transmit the file on a hop-by-hop basis, as opposed to breaking the files into fragments and routing each fragment, as in traditional routing protocols. To address this, we integrated fragmentation and network coding to work with DIRECT. This combination offers greater content-dissemination performance in networks with unreliable, short-lived paths.

3.1.2 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 “light weight” tag for node descriptions, and the “heavy weight” 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 heavy-weight content (see the Configuration Manual). We added a Flood module that is typically registered for light-weight content. It forwards content to all of the node’s neighbors, and relies on the Bloom filters to avoid routing loops and redundant transmissions.

Message Ferrying. As another generalization of DIRECT, we support two different types of flooding for node descriptions and arbitrary data objects: contemporaneous flooding with and without *push on contact*. This refers to the method of immediately forwarding a data object that is marked for flooding to each of a node’s neighbors (respecting the Bloom filters). *Push on contact* refers to a method of triggering subsequent contemporaneous floods upon discovery of a new neighbor. Specifically, when this option is enabled and a new neighbor is discovered, the discovering node will push all of the data objects marked for flooding to the new neighbor (which may trigger an additional flood to its neighbors). *Push on contact* is essential for supporting message-ferrying scenarios, where the subscriber and message ferry are not attached to the connected component of the data publisher at the time of publication.

Proactive Replication*.* We generalized ForwarderFlood to support flooding of arbitrary data objects, 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. A partial order is a binary relation that is:

1. reflexive (a <= a),
2. anti-symmetric (if a <= b and b <= a, then a = b);
3. transitive (if a <= b and b <= c, then a <= c).

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 in step 2 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 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.

3.1.3 Semantics of Haggle and First Class Applications

The mathematical semantics of Haggle query resolution is defined in [6]. In a nutshell, the relevance of a data object relative to a weighted set of interests (e.g., a node description representing the query) is computed by the weighted sum of overlapping attributes. Weights always refer to weights of interest attributes, not attributes of data objects (which are not weighted). The result of a query is the ranked set of data objects up to a given matching threshold and up to a certain maximum rank (limiting the number of results in the query). Weights and thresholds in Haggle are represented as integers in the range 0...100.

This is a fairly general and powerful concept that goes beyond traditional disjunctive and conjunctive queries. Unfortunately, Haggle does not implement this semantics consistently. Another complication is that Haggle uses two different variations of matching, one for applications (called filtering) and one for routing (called matching). Our experience is that in unmodified Haggle, only a disjunctive interpretation of interest (that is consistent with aggregation by union) is of practical use.

In ENCODERS, we modified Haggle to consistently use the above-mentioned 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 Configuration Manual for how to enable it and how to test queries.

3.1.4 1-Hop Bloom filters

Haggle uses Bloom filters (propagated as part of node descriptions) to avoid sending duplicate content. Specifically, before a node A sends a data object D towards another node B, A checks B’s Bloom filter for D in the node description in A’s data store. The combination of DIRECT with Bloom filters, as implemented in ENCODERS, is quite powerful, because unnecessary transmissions of large data objects can be blocked before they reach the neighbor of the requesting node. This is an approximation of our interest-driven routing algorithm developed for DTN [14], which used a separate knowledge-dissemination algorithm to exchange cache summaries.

Haggle implements multi-hop Bloom filters to reduce the number of matching data objects in a result set for a particular query. This reduction can save bandwidth when the queries are general and may match lots of content, but do not provide any benefits when the query is very specific (i.e., a hash of the data object's content): when a node receives the requested data object it simply unsubscribes. ENCODERS implements 1-hop Bloom filters for more efficient dissemination than flooding of Bloom filters through the Interest and Forwarding Managers.

3.1.5 Limitations and Possible Future Directions

Content dissemination in ENCODERS addresses several limitations of previous versions, but not all of them. There were no unforeseen problems with our approach. Indeed, we successfully integrated DIRECT relatively seamlessly with other complicated extensions, including network coding and fragmentation, demonstrating that the overall design appears to have a good modularity.

There are some inherent disadvantages related to the underlying transport protocols. Unmodified Haggle uses host-to-host TCP connections to exchange node descriptions. This approach will not be effective in wireless scenarios where a single broadcast packet in DIRECT corresponds to a plurality of packets proportional to the number of neighbors. This increase in the number of transmissions will increase channel congestion and packet loss, increase delay, and consume more power than a broadcast approach to interest propagation. As described in Section 3.2, we implemented a prototype of UDP broadcast of node descriptions to address this limitation.

Future versions could keep more node-description history details, instead of only the most recently received node description. The current version does not support hybrid routing or multiple Forwarding Managers for the same classification. One could extend this design to support a sequential application of Forwarding Managers and allowing multiple Forwarding Managers for a class of content. It might be beneficial to extend the classifier module to use the event queue, to enable delayed forwarding and aggregation. In this case, the classifier would post forwarding events that specify the classification, while the forwarding modules would register to these events (as opposed to the classifier calling the forwarding module directly).

As future work, one could explore the pub-match and stop-match semantics for our version of DIRECT. We believe that our architecture is general enough that adding these features will not be difficult, and it would be useful to evaluate their potential efficiency advantages in the context of CBMEN.

Proactive Replication*.* A key limitation to our approach of flooding as a means of proactive replication is its inherent wastefulness. Indeed, this approach cannot be used in scenarios with a large number of nodes, or where large data objects must be proactively replicated. One could explore a new replication mechanism that allows the user to specify the scope and frequency of the flood to address this limitation. Hybrid routing that is dependent on the scope of the message could be another solution: a data object can be proactively replicated within a small scope of the publisher, and routed using DIRECT outside that scope. Another interesting hybrid idea is reactive flooding, where the flooding is triggered only if some interest exists in the network. One may explore more advanced forms of proactive replication that are integrated with the caching architecture. Of long-term interest is the use of distributed hash table techniques for storing content in specific regions of the network as a means of increasing scalability.

Bloom Filters*.* The overhead and the implications of exchanging Bloom filters need to be carefully examined. Currently, a Bloom filter fills several packets. Each node description includes a Bloom filter, so any efficient broadcast-based dissemination of node descriptions must handle packet loss. Hence, a naive replacement of TCP by UDP may be insufficient if the loss rates are very high. On the other hand, if we remove the Bloom filters to reduce interest size, we will have to add a mechanism to prevent loops and a broadcast storm of node descriptions.

There are some inherent limitations of Bloom filters, which can be a problem for some scenarios. Bloom filters can quickly check set membership, however there is a small probability of a false positive. Conceivably, albeit highly unlikely, this situation could prevent a node from receiving important content indefinitely.

In Haggle, nodes use Bloom filters that represent their neighbor's cache state to make forwarding decisions. This state can become invalid if freshness is not carefully maintained. In certain situations, the unmodified version of Haggle prematurely sets these Bloom filters, which can lead to invalid knowledge propagation and worsen the effects of false positives. To alleviate this problem, we added a mechanism for periodically maintaining consistency with the counting Bloom filter. We also added configuration parameters to allow users to specify whether data objects should be removed from the Bloom filter when they are removed from the data store. Our long-term goal is a solution that will delete content from the Bloom filter, but the timing must be right to avoid content circulation and achieve time synchronization requirements.

The further apart nodes are in the network, the more out of date the Bloom filters will become. The more specific a query is (or the more unique a piece of content is) the less benefit there is from using Bloom filters. Bloom filters do not offer a clear advantage if an application that subscribes to a specific piece of content immediately un-subscribes upon receiving that content. Bloom filters do not generally prevent the duplication that occurs when a node first adds an interest for data.

Interestingly, Haggle computes hashes over the content, the attributes, and the creation timestamp, when computing the content identifier, which also serves as a Bloom filter hash. Thus, if two data objects have the same content but different attributes then they will both be sent. From an architecture perspective, this is necessary for correctness to prevent situations where nodes do not learn additional attributes for content due to a Bloom filter hit. This mechanism will not need to be changed to support the dissemination of content and metadata separately, because this can be done in separate data objects that are linked together, e.g., by a unique identifier.

In summary, 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.

3.2 Content-based Protocols: Lightweight Unicast and Broadcast

In this section, we describe how and why we 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.

3.2.1 Light-weight 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.

3.2.2 Limitations and Possible Future Directions

Limitations of the Initial Design*.* 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. We might explore increasing reliability passively by overhearing the receiver's retransmission. 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.

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.

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.

When implementing the proposed design, several unexpected problems arose with UDP and link-layer interactions, which led to poor performance. Specifically, our initial implementation had poor performance if the ARP cache entries between the communicating parties were nodes. Our solution was to exploit the fact that Haggle only uses single-hop host-to-host communication, thus a Haggle application has the necessary (IP, MAC) mappings before the ARP cache is populated. We added a mechanism to exploit this information by manually inserting these entries into the ARP cache[3]. This problem does not arise in TCP-only communication, as TCP has reliability guarantees and the kernel buffers the TCP packets until the ARP cache is full.

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.

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 can allows a future design where the protocol choice can be made at runtime, based on a range of factors and optimization objectives. Another practical approach to improve UDP broadcast is to use promiscuous mode, which could be naturally implemented as a hybrid between our UDP broadcast and unicast protocols. Another promising direction is to insert an additional layer above UDP, like NORM, which provides better reliability though forward error correction and optional (negative) acknowledgements.

3.3 Content-Based and Utility-Based Caching

Because content-based mobile edge networks can persistently store large amounts of content, a key concern is developing mechanisms that 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 content that is relevant to the user. Unlike IP-based networks, content-based networks maintain explicit information about the meaning and the use of content at a level of abstraction corresponding to semantically meaningful units that can be exploited for better content-management and in semantic networking.

The goal of content-based caching is to design efficient and effective content-aware caching algorithms inside the Haggle framework. We developed such algorithms that 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 together with proactive dissemination of content. When more content is flowing through the network, it becomes important to make fine-grained decisions of what content to cache and quickly discard content with insufficient utility. This decision needs to be based on the benefits and costs for each individual piece of content in a context-dependent and cooperative manner.

3.3.1 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

[3] For security reasons, we implemented this insertion process as a separate C program.

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[4] 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 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,

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

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), while FTP or web transfers do not necessarily have jitter or latency requirements.

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 the replacement component 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 also 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. For example, assume there are three different replacement modules R1, R2, R3, where R1 is the highest priority and R3 is the lowest priority, that are applicable to a data object with an attribute (A). The replacement module R1 with the highest priority will be checked first, to see if it is responsible for the data object. If the attribute (defined in the configuration file) is matched, R1 will handle the object accordingly (e.g., total ordering on the attribute A). Otherwise, it will proceed with the next highest priority (R2). Our sample configuration at the end of this section specifies replacement mechanism with priority policies based on mission timestamp and content create 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. As first step toward generalization of the ordering, we implemented lexicographical rules by using the aforementioned 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.

3.3.2 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. Our design extends the Haggle caching design, and in Phase 2 we implemented utility-based caching as a cache strategy in the existing design.

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).

3.3.3 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 there is a Blue-Force tracking application running on phones that 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 the network. 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 Blue-Force 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 the 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).

3.3.4 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.

3.3.5 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.

3.4 Network Coding and Fragmentation

Fragmenting files is a well-known and indispensible mechanism in peer-to-peer networking. Networking 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.

3.4.1 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.

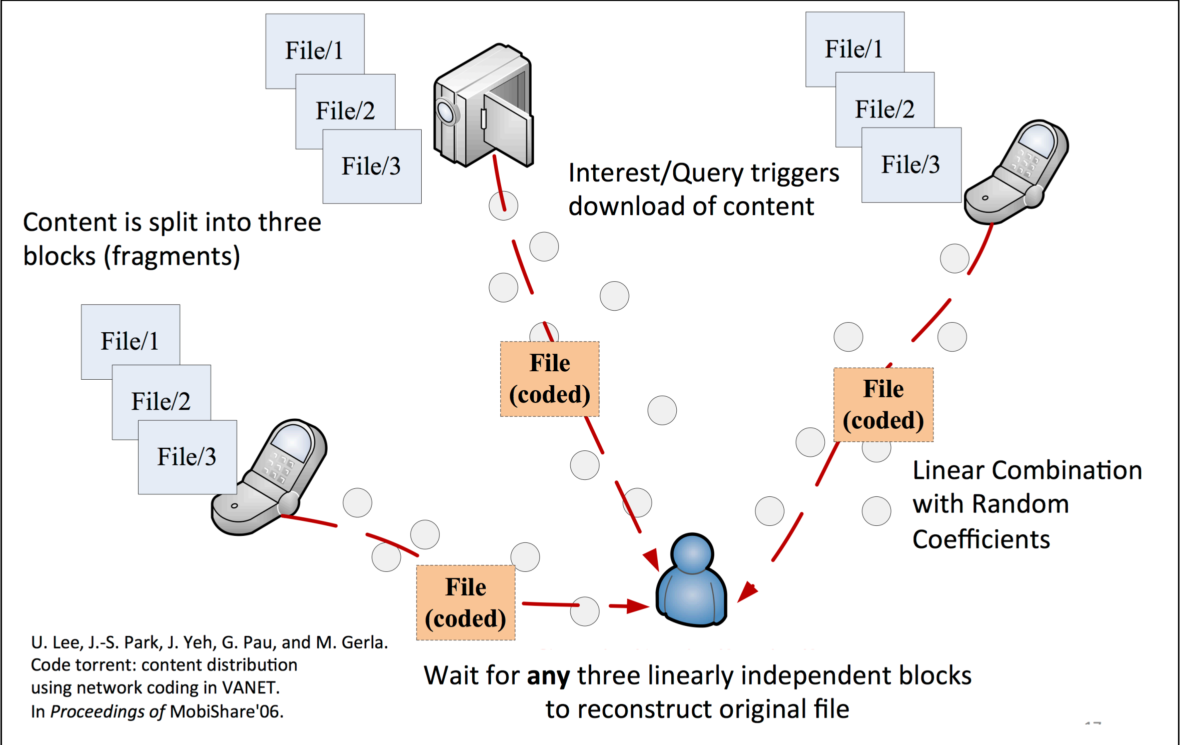
File/1
File/2
File/3
Content is split into three blocks (fragments)
Interest/Query triggers download of content
File (coded)
Linear Combination with Random Coefficients
U. Lee, J.-S. Park, J. Yeh, G. Pau, and M. Gerla. Code torrent: content distribution using network coding in VANET. In *Proceedings of* MobiShare'06.
Wait for **any** three linearly independent blocks to reconstruct original file

Figure 3. 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 approach to enhancing Haggle with network coding is based on an adaptation of Code-Torrent [1] to content-based networking with content caching and interest-driven content dissemination. **Figure** 3 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.

3.4.2 Network Coding Implementation

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.

Here, we describe the challenges that we overcame with our implementation of network coding in Haggle. 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.

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 in the implementation. Block data objects inherit all attributes of the data object being encoded. As described in the Technical Documentation, we include several additional attributes in all block data objects to provide information about the parent data object the block was derived from. That documentation also describes the classes, events and event flow for network coding and fragmentation.

3.4.3 Fragmentation Implementation

Fragmenting files is an indispensible 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.

3.4.4 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

$$l(d) > l_{thres}$$

where P_{thres} and l_{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 *l(d)* is the size of data object *d*. For data objects with size larger than l_{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.

3.4.5 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 adopts the solution of representing 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 uses a single generation for an object and a fixed generation size. This limits the content size to which we can apply network coding. This is mitigated by using fragmentation. ENCODERS fragments large data objects into multiple generations (as fragments are called in the context of network coding), 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 required 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 also 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.

3.5 Security

3.5.1 Overview

Figure 1 depicts the scope of the security solutions developed by SRI in ENCODERS within the context of the threats, features and existing security solutions.

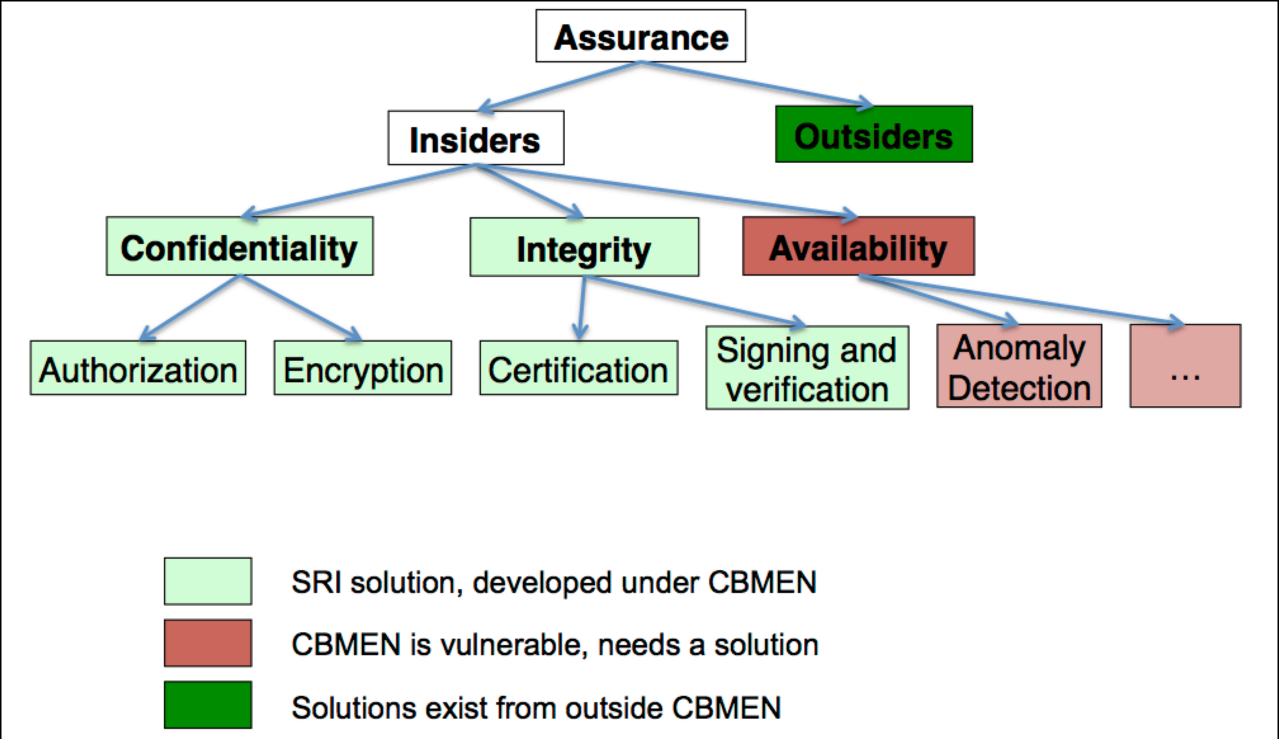

Figure 4. The security solutions developed in ENCODERS for confidentiality and integrity are shown in the context of the broader assurance tree.

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.

3.5.2 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))

3.5.3 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 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.

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.

In the following, we focus on the mission-specific aspects of security CONOPS: pre-mission security design and configuration; and security considerations during mission execution.

4.1 Pre-Mission Security Operations

Assume there is a mission database (MDB) that contains a list of mission personnel (warfighters, or WFs) and their mission roles/security levels. A mission-security officer (MSO) is responsible for designing the security space (attributes and grouping into roles) and making sure the interest attributes are compatible with security needs. A mission IT administrator is responsible for using the design to generate and deploy configuration files.

4.1.1 Pre-Mission Security Design

The MSO must determine security needs and then (1) design security attributes, and (2) rules for access control. A design tool (with access to the MDB) would guide the MSO to carry out the following tasks:

1. determine the certificate authorities (CAs)

2. determine the attribute authorities (AAs), their names and attributes managed, and the grouping of attributes into roles -- which may overlap

3. define relations:

    1. (wf,ca) -- wf (warfighter) can be certified by ca (in CA)
    2. (wf,aa) -- wf can request attributes from aa (in AA)
    3. (wf,aa,role) wf has access to role in roles from aa (in AA)

4. set the security level (what is signed, is access control on)

5. upload this information to the mission database

Notes:

1. Every WF should be related to at least one ca and one aa
2. There could be just one ca and one aa and they could be the same.
3. Every WF could be related to every ca and every aa
4. Roles are a mechanism to simplify access control and enable dynamic addition of attributes to a warfighters capabilities.
5. Possible extremes are all attributes in one role, and one attribute per role.

4.1.2 Configuring mission security

At the Forward Operating Base (FOB), before starting a mission, the Mission IT administrator uses a Mission Configuration Tool (MCT) to

1. define a mapping between ids of available device and WFs
2. store the mapping in the mission db.
3. generate configuration files for each WF (the assigned device) from information in the mission database, including the security design
4. load the configuration files onto the assigned devices. (They may be stored in the db and loaded from there).

The configuration file has a section for each Haggle manager. Here we only discuss configuration of Security Manager (SM), which uses the relations defined by the MSO. Detailed discussion of the security configuration parameters can be found in the ENCODERS Configuration Manual. The main tasks for configuration are:

1. decide appropriate values for security parameters (signing, encryption, security level, etc)
2. design an appropriate model for encryption attributes in the scenario (see examples below)
3. create a JSON specification for the above
4. run the security configuration tool to generate configuration documents for each node
5. upload the configuration documents

Start up. When ENCODERS starts up, nodes request and receive signing certificates from their associated CAs, and request and receive encryption and decryption keys from their AAs. Everyone gets all encryption keys, as they correspond to the public keys in a standard public-key system. There are several choices for distribution of decryption keys.

1) *by need*—a node only requests a decryption key when first needed (content is received encrypted using a policy that requires the decryption key).

2) *all allowed*—a node requests all the decryption keys that it is assigned at startup.

3) *all*—when a node requests decryption keys it is given those it is assigned, and an additional package with additional keys encrypted so they can only be accessed by use of a suitable password.

The *all* distribution mode allows for role/status change during a mission, while avoiding the need to reconfigure an authority (assuming the authority is available).

4.2 Security issues during mission execution.

We assume that the device assigned to a 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 for the following situations:

- Encryption of data in ENCODERS devices
- ENCODERS device requires a reboot -- hangs, battery falls out
- ENCODERS device needs to be replaced
- Warfighter role is changed

- An unauthorized user accesses the network using an ENCODERS-enabled device
- ENCODERS devices operating as Data Couriers or Delegate Nodes

4.2.1 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.

4.2.2 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 transmitted (and cached) by other nodes.

4.2.3 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.

4.2.4 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.

4.2.5 Unauthorized user accesses the network

We assume the user has an ENCODERS-enabled device. Nevertheless, the user may will 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.

4.2.6 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 can not deliver any decrypted content to applications on the node. They subscribe to content that they should carry or forward.

4.3 Security Design Examples

In this section, we present a set of examples of the use of ENCODERS security in different scenarios.

4.3.1 Scenario 1

This scenario has one squad with three units on a small cordon and search mission (see Phase I Drexel scenario). Units U1 and U2 are clearing east/west access to a village, and U3 will then search the village center for a HVT.

Authorities: one authority A1 (for certificates and security attributes). Thus, we have (WF,A1) in the "can be certified by" relation and in the "can request attributes from" for each WF.

Attributes: who has access:

1. A1.SQ1 -- everyone in squad 1
2. A1.U1 -- everyone in unit 1
3. A1.U2 -- everyone in unit 2
4. A1.U3 -- everyone in unit 3
5. A1.SQL -- every squad leader that knows A1

There are 4 roles:

1. A1.SQL1 = {A1.SQ1, A1.SQL} -- squad1 leader (SQL1)
2. A1.Unit1 = {A1.SQ1,A1.U1} -- WF in U1
3. A1.Unit2 = {A1.SQ1,A1.U2} -- WF in U2
4. A1.Unit3 = {A1.SQ1,A1.U3} -- WF in U3

At the start of the mission, SQL1 publishes a picture of the HVT with access (A1.SQ1 and A1.U3) and attributes -- *image, hvt*. The picture's metadata contains the HVT name, but this is not a visible/matchable attribute.

A WF in U2 takes a picture of a suspicious person and publishes it with access (A1.SQL1 or A1.U3) and attributes *image, suspicious.* The encrypted image is annotated with time and location.

The image matches the picture previously sent to U3. Now they know which way to go. U1,U2 WFs may have the two images cached, but do not have the decrpytion keys, they are not in role SQL1 or Unit3. Thus if a terrorist gets access to a U1 or U2 phone, but no information about the hvt is leaked other than the interest in HVTs and suspicious things.

4.3.2 Platoon level scenario (Phase I VIP Demonstration)

The platoon is on a security patrol, looking for IED, suspicious people, talking to villagers. In this area, a platoon is also prepared for cordon and search, either for rescue or capture of a target.

Authorities: One Certificate Authority (CA) and One Attribute Authority (AA). If they are not the same node, there is less congestion at startup time.

Attributes and intended meaning:

1. AA.Lds—leaders PL1 PL2 SL1 SL2 SL3 CC
2. AA.IED—access to IED information and sightings
3. AA.Tgt—information about people, including HVTs, suspicious activities, village leaders
4. AA.P1—anyone in Platoon 1
5. AA.SQ1—anyone in squad 2
6. AA.SQ2—anyone in squad 2
7. AA.SQ3—anyone in squad 3

Types of content and who has access:

1. GPS from P1 to P1
2. General mission data to P1
3. Details such as routes and rally points to Lds
4. Specific HVT info to LDS + Tgt
5. Sensitive spot reports to LDS or just PL1, CC
6. SitRep from P2 to Lds -- may be republished with different access
7. Rescue plans, diagrams -- Search Role

Roles for Rescue mission:

1. AA.Cordon—secure perimeter for rescue/capture mission
2. AA.Scout—gather situation information at target site
3. AA.Search—carry out rescue/capture

4.3.3 Two Platoons passing

A CA and an AA for each platoon

1. CA1 certifies all P1 nodes and bridge nodes in P2
2. CA2 certifies all P2 nodes and bridge nodes in P1
3. AA1 provides attributes to all P1 nodes
4. AA2 provides attributes to all P2 nodes

PL1 is coming in from patrol, PL2 is going out. PL1 has content that can be accessed by PLT leaders with policy: AA1.PL1 OR AA2.PL2 and content that can be accessed by SQLs with policy: AA1.SL1 OR AA1.SL2 OR AA1.SL3 and content available to anyone in the company with policy: AA1.P1 OR AA2.P2

How do we design attributes/certificates so this works? Content can flow from PL1 via bridge nodes in P1 through bridge nodes in P2 to PL2. Content access is scoped by the above policies.

4.4 API

The ENCODERS API is essentially the Haggle API. The security component mainly concerns setting a policy attribute on data objects containing content to be published.

add_attribute(dobj,"policy",policystring)

where policystring is (using java syntax) either a single attribute

"Authority1.Attribute1"

a disjunction of policies

"(" policystring1 + "OR" + policystring2 ")"

or a conjunction of policies

"(" policystring1 + "AND" + policystring2 ")"

The outermost ()s can be omitted. For example

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

Applications should be parameterized by the specific set of attributes available and the rules for assigning policies to content. Additionally, applications are provided with functionality to dynamically change some of the security parameters.

Applications can dynamically change the security configuration on a node without requiring a reboot of the haggle kernel. This is accomplished through the use of various new API calls that can change security parameters. Specifically, the following can be changed dynamically:

1. node shared secrets
2. role shared secrets
3. list of configured authorities at a node
4. list of nodes authorized for certification at an authority
5. list of roles authorized for attributes at an authority

5 Integration and Performance

Integration was facilitated because SRI's design used an existing open-source framework. From project inception, all members of the SRI team and the MSI's team were able to proceed in parallel to meet CBMEN Program milestones. SRI's Technical Documentation describes our use of a git-based workflow and branches to support maximally concurrent development activities.

In most cases, integration boiled down to a simple git-merge. This turned out to be the case for the integration of dissemination and caching. It also worked in the case of dissemination/caching and network coding. Only a few minor problems were detected and quickly fixed. The integration with fragmentation was similarly straightforward. The integration has been further improved by disseminating metadata proactively and optionally through broadcast instead of relying on default interest for such data at each node. The integration of signing with network coding and fragmentation required some additions to pass through signatures.

The integration between network coding and fragmentation was possible with hardly any changes in the components, which was surprising. Furthermore, it was instructive to see that it was possible to add UDP-based protocols, even broadcasting for node descriptions, without changes to the routing architecture.

SRI is maintaining a semantics branch, which cleans up the Haggle semantics to bring it closer to its mathematical specification. This branch took more effort to integrate and test because it affects various parts of the system, but in the end it flawlessly worked together with all other components.

The encryption branch extends the basic security branch with multi-authority certification and encryption. The attribute-based encryption primitives are implemented by embedding the Python interpreter inside the Haggle process and using that to invoke the CHARM library. The integration of encryption with the other features (especially fragmentation and network coding) was non-trivial, but our modular design made it possible without major problems.

Regarding the Haggle framework, our experience is quite positive. We identified and documented a couple of limitations and quirks for our team members, and found and fixed a number of bugs. We had few issues during the development, and our developers were able to quickly master the complexity of the framework and the new components because of the existing documentation and frequent SRI-team discussions, which were facilitated by hosting other team members as visitors at SRI. 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 more time-consuming.

The ease of integration allowed us to focus on the difficult problems, such as the right architectural decisions and algorithm design to allow maximum flexibility, including mix and match of different components or features and full parameterization through configuration, which in the future can become more dynamic and adaptive.

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. Our Final Reports for each phase summarize our performance results.

6 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, 2009.
7. —. "Haggle: Relevance-Aware Content Sharing for Mobile Devices Using Search." 2012.
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.