Usage Management in Information-centric Networks

Gregory Heileman and Christopher C. Lamb

Department of Electrical and Computer Engineering
University of New Mexico

September 2012

UNM | Electrical & Computer Engineering

Introduction

1. Information-centric Networking
2. System Overview
3. Theoretical Foundations
4. Implementation Overview
5. Final Steps

Information-centric Networking

Information-centric networking (ICN) is a new approach to internet-scale networks. These networks take advantage of data locality, cache data aggressively, decouple information providers from consumers, and use a content-centric perspective in network design

Similar conceptual approaches:

- **Named Data Objects** — Data objects are the primary data abstraction.
- **API Structure** — Programming interfaces are structured around requesting specific data objects. Can be synchronous or asynchronous.
- **Naming and Security** — Names are tightly and securely bound to content.
- **Caching** — Content is aggressively cached on nodes.

We are most concerned with the first and second characteristics currently.

Current Technology Fail

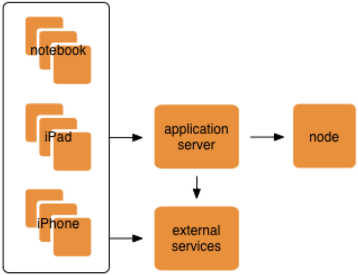
Current internet technologies don't dynamically protect data. Initial design assumtions and implementation characteristics make these approaches difficult.

Why does the internet fail?

- **Strict Layering** — Routing and switching are generally lower level operations (layers 2 and 3 in the OSI model). Content-sensitive routing is a layer 7 (application layer) operation, requiring very expensive hardware to do well.
- **End-to-End Arguments** — Network cores are simple, fast, and dumb. They need to brighten up to evanluate information suitability.
- **Packetization** — Policies and content requires multiple packets. This implies complex window retention logic to handle context splitting, and in at least half of typical cases, context splitting can't be handled at all.

The principles have been effective, but in some cases services *should* be in the network core.

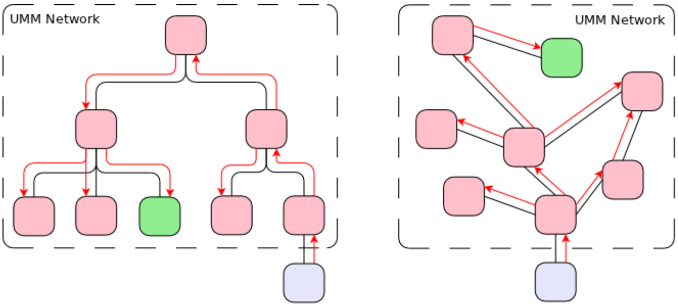
System Overview — Device Perspective



How does the system work?

1. **Request** — An initial request is submitted from an edge device
2. **Reciept** — The request is received by an application server that has access to ICN services via an ICN node and external services of some kind as well.
3. **Dispatch** — The request for information if suitable is dispatched into the ICN.

System Overview — Into the network

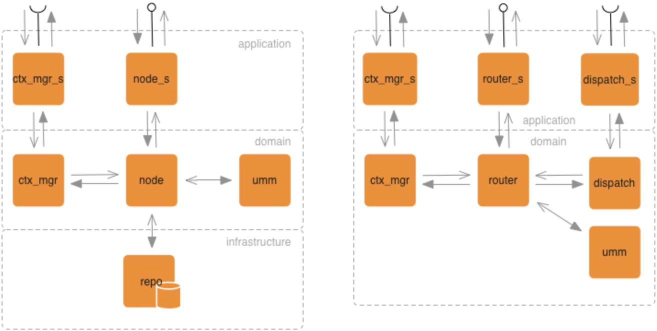


(A) Hierarchical Network (B) Non-hierarchical Network

How is information dispatched through the network? *Well, it depends on the network.*

1. **Submission** — Submit to a node.
2. **Transmission** — Pass through the network, either via a router (if hierarchical) or known peers (non-hierarchical).
3. **Respond** — Nodes respond with content if possible.

System Overivew — Nodes and Routers



(a) Hierarchical Node

(b) Hierarchical Router

Strictly layered architecture within nodes:

- **Application** — REST Interfaces, network access
- **Domain** — Domain functionality, specific node/router logic and UM
- **Infrastructure** — Data storage repositories, data objects and policies

Distributed Security Decisions

In order to make security decions, we must know
the *environment* of the *resource* (the data object) and the *subject* (the user).

Fundamental Foundations

In order to understand that our decisions are valid and maintain security in distributed *environments*, we must *prove* that local security decisions provide a compliant global solution.

We make *greedy* with respect to security, and for efficiency we use *dynamic programming*. For these to provide a globally optimal solution, we need to show that the routing problem exibits *optimal structure* and *overlapping subproblems*.

Optimal Substructure

Idea: If we then have a path P consisting of nodes and edges $\{V, E\}$, that path was assembled by choosing the most secure edge $e \in E$ from a corresponding node $v \in V$ at some time t. The path P consisting of these edges e is then the most secure path that can be chosen.

Proof by Contradiction

1. Assume a path $P = \{V, E\}$ chosen using a greedy algorithm.
2. Assume that P is not the most secure, and that a more distinctly secure path $P' = \{V, E'\}$ exists.
3. If P' exists, then at some $v \in V \ \exists \ e' \in E$ such that e' is more secure than the corresponding edge $e \in E$.
4. If so, then at all$v \in V$, $e' = e$ leading to $E' = E$ and $P' = P$, so P' is not distinct from P.

This is essentially the shortest path problem. In this case possible solutions are constrained however, because we don't have access to information about the entire network. We can only evaluate context at each node.

Overlapping Subproblems

Idea: We can caluculate a context to determine the security posture of a given edge. If the parameters involved in calulating that value don't change, then the security value of that edge won't change either.

Proof by Contradiction

1. Assume a security value v associated with an edge $e \in E$ calucuated via contributing parameters $p \in P$ by an idempotent security function f.
2. Assume $\exists$ an alternative security value v' for e calculated from the same parameters p by f.
3. Then the function f returns different outputs O based on the same input p.
4. $\nexists$ v' due to the definition of idempotence.

The security functions are deterministic and don't incorporate any random behavior, so we can assume idempotence as a characteristic.

Implementation Overview — Environment

The simulation environment runs over a nation-spanning overlay network hosted via Rackspace Cloud and Amazon Elastic Compute Cloud (EC2). The network uses ICN concepts in interface design and operational semantics.

Overlay Technical Components

Category	*Components*
Infrastructure	Amazon S3, Amazon EC2, Rackspace Servers
OperatingSystems	Ubuntu 11.04, Ubuntu 12.04
Technologies	Ruby (Sinatra, Capistrano, YAML), REST
SupportingSystems	Git, Github

Implementation Overview — Flow

We examined two different interation architectures:

- *Cat Model* — Similar to a filesystem using a command-line interface to access content.
- *Index Model* — Akin to typical web-based systems using an index file referring to other content.

We chose the *Index Model*. But why?

- *Content Listing* — Content listing on an arbitrary network requires hop count or duration limits in order to maintain responsiveness.
- *Content Listing* — Precedent for this system in this context (BitTorrent, etc.).
- *Content Listing* — Focus of work is web-centric, so a web-centric approach is a more natural fit.

Implementation Overview — Usage

Planned execution domains can have policies over five different dimensions. We have three separate domains with different policy taxonomies.

Dimension	*Type*	*Required?*	*Domain A*	*Domain B*	*Domain C*
Affiliation	Set	Yes	tropic_thunder, gallant_entry	tropic_thunder, gallant_entry	tropic_thunder, curious_response
Sensitivity	Ordering	Yes	unclassified, secret, top_secret	unclassified secret, top_secret	unclassified, secret, top_secret
Category	Set	No	aqua, magenta, vermillion	alpha, beta, gamma	one, two, three
Organization	Set	Yes	Oceania, Eastasia, Urasia	Oceania, Eastasia, Urasia	Oceania, Eastasia, Urasia
Device	Set	No	workstation, tablet, phone	workstation, phone	workstation, tablet

Implementation Overview — Index

Initial content is located via an index.

Listing 1: Seed Information for the Network

```
<index>
  <location>
    <name>The location name; a city name, for example</name>
    <lat>The location latitude</lat>
    <lon>The location longitude</lon>
    <about>Metadata about the location</about>
    <key>The detail data object name</key>
    <key>...</key>
    ....
  </location>
  <location>...</location>
  <location>...</location>
  ...
</index>
```

Implementation Overview — Data

Currently, we process images, shapes, and generic content.

Listing 2: Image

```
<artifact>
  <policy-set>
  ...
  </policy-set>
  <data-object>
    <image type=
          >
    ...
    </image>
  </data-object>
  ...
</artifact>
```

Listing 3: Shape

```
<artifact>
  <policy-set>
  ...
  </policy-set>
  <data-object>
    <shape type=
          >
    ...
    </shape>
  </data-object>
  ...
</artifact>
```

Listing 4: Content

```
<artifact>
  <policy-set>
  ...
  </policy-set>
  <data-object>
    <content type
        =   >
     ...
    </content>
  </data-object>
  ...
</artifact>
```

Implementation Overview — Shapes

We hande a variety of shapes.

Listing 5: Marker

```
...
<shape type=
                >
  <marker>
    <lat>...</lat
       >
    <lon>...</lon
       >
  </marker>
</shape>
...
```

Listing 6: Circle

```
...
<shape type=
                >
  <center>
    <lat>...</lat
       >
    <lon>...</lon
       >
  </center>
  </radius>...</
     radius>
</shape>
...
```

Listing 7: Polygon

```
...
<shape type=
                >
  <vertex>...</
       vertex>
  ...
</shape>
...
```

Implementation Overview — Policy

Listing 8: Policy DSL Example

```
policy_set {
  policy(:p1) {
    match :all
    rule(:mission_affiliation) { |x| x == :tropic_thunder }
    rule(:sensitivity) { |x| x == :top_secret }
  }

  policy(:p2) {
    include :p1
    match :all
    rule(:device) { |d| d == :workstation || d == :phone }
  }

  policy(:p3) {
    include :p1
    match :one
    rule(:category) { |c| c == :vermillion }
    rule(:organization} { |o| == :oceania }
  }
}
```

Final Steps