Na Kika: Secure Service Execution and Composition in an Open Edge-Side Computing Network

Robert Grimm, Guy Lichtman, Nikolaos Michalakis, Amos Elliston, Adam Kravetz, Jonathan Miller, Sajid Raza
New York University
Dynamic Content: Easy to Build, Hard to Scale

- Dynamic content is increasingly popular, easy to create and publish
- Example: mash-ups
  - chicagocrimes.org = crime reports over Google Maps
  - zillow.com = real estate stats over Microsoft Virtual Earth
- Easy to realize on a home server
  - PHP, Python, ASP, JSP, ...
- Collaborate, plug together
- Does not scale
Requirements for a Collaborative Architecture

- **Scalable**
  - Move content creation and transformation to the internet's edge
    - Computing power and bandwidth to serve local clients
  - Support incremental growth, make it easy to administer

- **Easily extensible and composable**
  - Support dynamic mixing, mashing, melding

- **Secure**
  - Control access to (resources of) architecture
  - Control access to outside content
We Need a New Delivery Platform

- Clusters amplify resources, not necessarily near the client
  - [TACC, Veritas, Linux-HA]

- Edge-side hosting targeted at big trusted sites
  - [Akamai, ACDN, ColTrES, Tuxedo, vMatrix, WebSphere]

- P2P collaborative architectures limited to static content
  - [Coral, CoDeeN, CobWeb]

- Some efforts provide containment but not composition
  - [Active Cache, SDT]

- *Na Kika* reconciles extensibility with security
Talk Outline

- Introduction
- Architecture and programming model
- Design space
- Experimental evaluation
- Limitations and on-going work
- Conclusions
**Na Kika Architecture**

- DNS redirects clients to nearby proxies
  - Add “.nakika.net:7070”
- Proxies organized in structured overlay for caching static content
- Sites publish scripts just like other content
- Scripts are executed and composed on proxies to scale dynamic content
Na Kika Architecture

- DNS redirects clients to nearby proxies
  - Add “.nakika.net:7070”
- Proxies organized in structured overlay for caching static content
- Sites publish scripts just like other content
- Scripts are executed and composed on proxies to scale dynamic content
Write scripted code
- Easy to write, already familiar
- JavaScript

Structure functionality into event handlers
- onRequest handler
- onResponse handler

Specify pair of handlers through a Policy object
Service Modularity

- Leverage descriptive nature of HTTP messages
  - URL
  - Client IP address
  - Method
  - Headers
- Select handlers based on HTTP message properties
- Execute the most specific match

```javascript
var p1 = new Policy();
p1.url = ["*.zillow.com/*"];
p1.client = ["0.0.0.0/0"];
p1.method = ["GET"];
p1.onRequest = function(){ ... }
p1.onResponse = function(){ ... }
p1.register();

var p2 = new Policy();
p2.url = ["*.zillow.com/*"];
p2.client = ["128.122.0.0/16"];
p2.method = ["GET"];
p2.onRequest = function(){ ... }
p2.onResponse = function(){ ... }
p2.register();
```
Service Composition

- Pair of event handlers mimics proxy structure
- A series of handler pairs forms a *pipeline*
- Handlers composed via the `nextStages` property

```javascript
var p = new Policy();
p.nextStages = ['chicagocrime.org/map.js', 'cityofchicago.org/police/crime.js'];
p.onRequest = function() { ... }
p.onResponse = function() { ... }
p.register();
```
Admission & Emission Control

- Reuse same mechanisms
- Handler selection
- Composition
- Insert two extra pipeline stages
  - ClientWall near client
  - ServerWall near server
- Make security policies extensible

```javascript
var p = new Policy();
p.method = ["GET", "POST"]; p.onRequest = function() {
    Request.terminate(ACCESS_DENIED);
} p.register();
```
Containing Hosted Code

- **Scripts are sandboxed**
  - Select native libraries (vocabularies)
  - Accessing HTTP messages and state (cookies, cache)
  - Processing content (XML, images)
  - Otherwise, no direct access to system

- **Resources are only limited under congestion**
  - Insight: *Na Kika* already shared, public service
  - If no congestion, do nothing
  - Otherwise, throttle requests
  - Terminate largest consumers as a last resort
Talk Outline

- Introduction
- Architecture and programming model
- Design space
- Experimental evaluation
- Limitations and on-going work
- Conclusions
Five Challenges and Our Solutions

- **What interposition points to expose?**
  - onRequest and onResponse

- **What code to execute?**
  - JavaScript

- **How to specify and apply predicates?**
  - Policy objects specify predicates, are applied on requests

- **When to perform access control?**
  - ClientWall, ServerWall

- **How to perform resource control?**
  - Congestion-based
    - “Use as much as you need, but don’t cause overload”
What Interposition Points to Expose?

- Goal: Equivalent support
  - Content creation, content transformation
  - Security policy enforcement

- On responses only (Active Cache, SDT, TACC)
  - No support for access control, request redirection (for layering)

- Input filters, content handler, output filters (Apache, Java Servlets)
  - Less flexible: either transform or create

- Before and after cache, on storage access [Pai et al. '03]
  - Harder to compose, but more control & better performing
What Code to Execute?

- **Native code**
  - Provides little constraints on language, programming model
  - But is hard to contain, locks in hardware platform

- **Java byte code (and other intermediate execution formats)**
  - Is pretty flexible
  - But also is pretty hard to contain, requires complex runtime

- **Scripts**
  - Tend to be popular, simpler, more immediate (no compilation)
  - JavaScript: widely used, C-like syntax, prototype-based obj. model
    - Other scripting languages could be supported as well
How to Specify and Apply Predicates?

- **Specification:** Domain-specific language
  - Little gain in expressivity
    - Notably, top-level disjunctions
  - More complex to implement *and* use
    - Based on our experiences with prototype

- **Application:** Separately on requests and responses
  - More flexible
    - E.g., match on content type
  - Less predictable
    - onRequest and onResponse handlers are not matched
  - Already implemented, but not exposed to users
When to Perform Access Control?

- **ClientWall**
  - Strictly necessary to control admission into our system

- **ServerWall**
  - Necessary because of request redirection (for layering)

- What about scripts, explicitly accessed resources?
  - Security requires redirection through our architecture
    - But indiscriminate redirection can lead to infinite recursions
  - Administrative control stages have direct access to web
  - Site-specific stages only go through ClientWall & ServerWall
    - Can request regular script processing
How to Perform Resource Control?

- Based on quotas
  - Relative to content size (Active Cache)
    - Hard to set precisely for response processing
    - Impossible to set for request processing
  - Specified through predicates (extended policy objects)
    - Provides more flexibility
    - But amplifies administrative overhead
- In general: Any form of reservation is wasteful
Talk Outline

- Introduction
- Architecture and programming model
- Design space
- Experimental evaluation
- Limitations and on-going work
- Conclusions
Experimental Evaluation

- Are throttling and termination effective?
  - Both under overload and with malicious scripts

- How does *Na Kika* compare to a single server?
  - Wise-MD web-based learning application

- How easy is it to extend functionality/security?
  - Examples of *Na Kika* extensions
Implementation

- Leverages several open source packages
  - Apache as web runtime
  - SpiderMonkey as JavaScript runtime
  - Coral as structured overlay
  - OASIS for DNS redirection

- Relies on processes to handle concurrent requests
  - Simplifies resource accounting, pipeline termination

- Runs scripts in user-level threads
  - Hides Apache's API for piecemeal processing of HTTP messages
Implementation (cont.)

- Exposes underlying buffer data
  - By adding byte arrays as new core data type

- Makes extensive use of caching
  - Content including script sources
  - Lack of scripts
  - Compiled scripts
  - Decision trees (representing collection of policy objects)
Resource Controls Can Be Effective

- Throttling rejects less than 0.55%
- Terminating drops less than 0.08%

With resource controls

- High Load
- Higher Load
- Misbehaving

Without RC

With RC
Wise-MD

- Web-based education tool developed at NYU medical school
  (formerly known as SIMMs)
- Global participation
- U.S. + Australia
- Multimedia intensive
- 1 GB total content
- Dynamic
- HTML generated from XML and XSL stylesheet
Wise-MD on Na Kika

- 1 developer, 100 + 130 lines of code, 2 days to port
- Single server compared to single, cold Na Kika proxy
- 160 clients replaying Wise-MD logs

<table>
<thead>
<tr>
<th></th>
<th>Single Server</th>
<th>Na Kika Proxy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HTML</td>
<td>Movies</td>
</tr>
<tr>
<td>Local network</td>
<td>904ms</td>
<td>100%</td>
</tr>
<tr>
<td>80ms delay, 8 Mbps cap</td>
<td>8.88s</td>
<td>26.2%</td>
</tr>
</tbody>
</table>

90 %ile latency for dynamic HTML, fraction of clients with ≥ 140 Kbps bandwidth for movies
Wise-MD on PlanetLab (movies)

- Single server compared to 12 Na Kika proxies
- 240 clients on 12 additional nodes replaying Wise-MD logs

![Bar chart showing fraction of clients in % for Single Server, Na Kika Cold, and Na Kika Warm categories with different data ranges for ≥ 140 Kbps, < 140 Kbps, and Failed.]

- Single Server: Failed 100, ≥ 140 Kbps 75, < 140 Kbps 25
- Na Kika Cold: Failed 75, ≥ 140 Kbps 50, < 140 Kbps 25
- Na Kika Warm: Failed 50, ≥ 140 Kbps 75, < 140 Kbps 25
Wise-MD on PlanetLab (dynamic HTML)

![Graph showing performance comparison between warm and cold caches with 240 Clients Na Kika and a single server.](image)

- **Warm Cache**
- **Cold Cache**
- **240 Clients Na Kika Warm**
- **240 Clients Na Kika Cold**
- **240 Clients Single Server**
Extensibility in Action

- Na Kika Pages (NKP)
  - Programming model similar to PHP, JSP, ASP

- Image transcoding
  - Transforms images to JPG, scales them down

- Annotated Wise-MD
  - Layer electronic post-it notes over Wise-MD

- Content blocking
  - First additional stage creates policy based on blacklist
  - Second new stage executes policy, rejecting illegal URLs
Annotated Wise-MD In Action

Module Selection:

**Adrenal Adenoma:**
In this module, you will learn how to work up an adrenal adenoma and gain comprehension of the complex physiology of the adrenal gland.

**Carotid Stenosis:**
In this module, you will learn about cerebral vascular occlusive disease and its relation to symptoms.

**Cholecystitis:**
In this module you will be introduced to the pathophysiology of Acute Cholecystitis and other diseases related to Cholelithiasis.
Extensibility in Action

- Na Kika Pages (NKP)
  - Programming model similar to PHP, JSP, ASP

- Image transcoding
  - Transforms images to JPG, scales them down

- Annotated Wise-MD
  - Layer electronic post-it notes over Wise-MD

- Content blocking
  - First additional stage creates policy based on blacklist
  - Second new stage executes policy, rejecting illegal URLs
Easy to build, Easy to Scale

- Less than 100 lines of code for each application
- Annotations relied on 180 lines of external code
- Less than 8 hours to write and debug
- Deployment at the edge scales
Talk Outline

- Introduction
- Architecture and programming model
- Design space
- Experimental evaluation
- Limitations and on-going work
- Conclusions
Limitations & On-going Work

- Source code must be public
- Sites gain capacity, but lose control over performance
- Unsuitable for applications with large databases
  - Basic hard state replication in place
    - Already running modified SPECweb99 [NSDI ’06]
  - Exploring better (partial) replication strategies
- Proxies are trusted
  - Protection against misbehaving/malicious proxies
- Resource management as congestion control
  - General technique for shared services
Conclusions

- *Na Kika* scales dynamic content
  - Focus on collaborative efforts

- We make two major contributions
  - Same mechanism for defining functionality and policies
    - Easy to program, easy to extend
  - Congestion-based resource management

- Proxies are crucial to extending large distributed systems
  - ACDN, Active Cache, CobWeb, CoDeeN, ColTrES, Coral, *Na Kika*, SDT, TACC, Tuxedo, WebSphere Edge Server, ...
  - Jeff Mogul: “When designing a new protocol, start with the proxy”
www.nakika.org