Software Design Specification

02/18/2010

Andrew Arminio
Christopher Austin
James McCauley

Introduction

While home and small office/home office networks have provide any number of benefits, they seldom live up to their potential. For example, security in these networks is often misconfigured, with at least two negative results. First, intruders can get into the networks – stealing identities, contributing to the amount of spam on the internet, and generally causing problems. Second, security misconfiguration limits actual network usability for legitimate purposes, which may make it easier to use a USB flash drive or an email to transfer a document within the same house rather than use the file and printer sharing features built into consumer operating systems. Beyond security, such networks often have implicit decisions which are not optimal for home users. For example, these networks provision bandwidth in a simple-minded “fair” manner. However, this is almost never the correct solution for the home. In the home, audio/video data (such as telephony and streaming music or video) and interactive web traffic should basically always be given preference to long downloads and peer-to-peer filesharing. Other such problems abound. In short: home networks do not serve home network users as well as they might.

Our client, Nicira Networks, is the primary developer of NOX, as well as the primary developer of Open vSwitch, which is a virtual network switch that, among other things, implements OpenFlow. These two technologies – NOX and OpenFlow – both stem from a project under the umbrella of Stanford University’s Clean Slate Design for the Internet called Ethane. NOX and OpenFlow work hand in hand with the aim of separating low-level packet forwarding from high-level decision making. Said another way, they make the network itself programmable, which enables smart networks.

Nicira's focus is and continues to be on on applying NOX and OpenFlow technologies to campus, corporate, enterprise, datacenter, and virtual networks. However, they also see that these technologies

may be of benefit to home and small office/home office networks. While there is more potential for NOX and OpenFlow in the home network than our team could possibly develop in a semester, we seek to provide a compelling proof of the concept, and to provide the proper "jumping off point" from which future home/SOHO networking tools can be built. This means we must cater to two separate audiences: the technical audience of people wishing to do network research or develop specific network "applications" for the home, and nontechnical consumers who simply wish to take advantage of these tools.

We have identified a number of principles – largely borrowed from Ethane – to guide our development. Briefly:

- **The network should be governed by policies declared over high-level names.** User and machine names like “Perry”, “The Kids”, and “Susan’s Dell” are far more valuable to home and SOHO users than lower level names such as MAC and IP addresses.
- **Network routing should be policy-aware.** User configurable policy should control network traffic – what gets sent where and when.
- **The network should track bindings between network traffic and the high-level entities that are its source and destination.** We need to be able to associate traffic with high-level names in order to correctly make policy decisions for that traffic.
- **An ordinary consumer end-user should be able to operate the system with a minimal investment of time.** While we hope that the use of high-level names will be a great step in this direction, this also very clearly calls for a graphical user interface that leverages the investment that users have already made in operating personal computers.
- **Users should not find the system to be a burden to use.** We must aim to make whatever improvements we can while impacting user experience as little as possible – home/SOHO users simply will not “put up with” a system that consistently impedes their usage or even reminds them it is there.

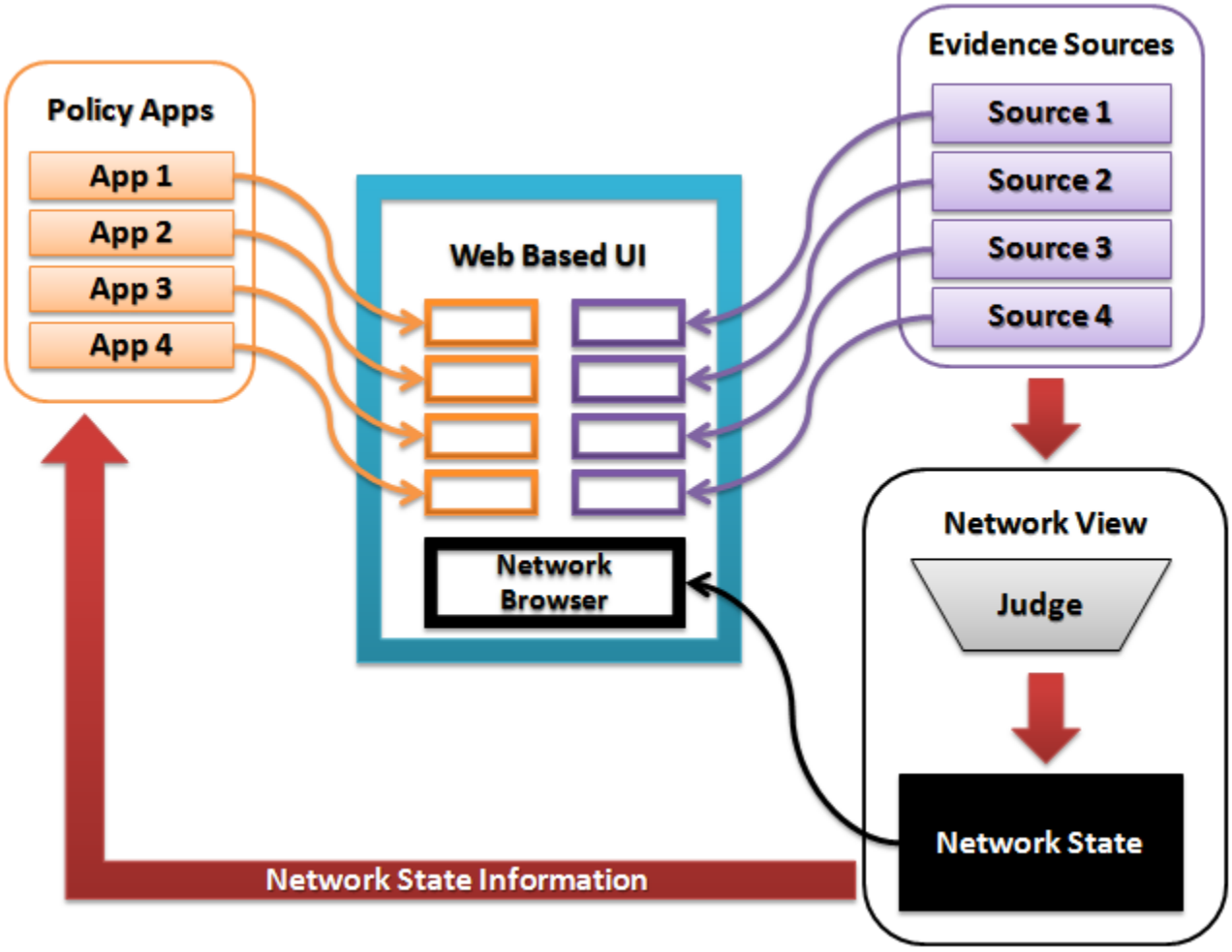
These high level goals motivate the development of a framework that enables the development of tools to improve home/SOHO networking. However, a framework based purely on high level goals is both difficult to design and difficult to evaluate. To this end, we have identified and will build some prototypical home/SOHO network tools. The specific features we have identified are:

- Identify machines by MAC address
- Identify users using passive methods (snooping on Instant Messaging and website account logins)
- Identify users actively -- through an explicit web login/logout page
- Maintain and display a view of the network -- which machines are online, and which users are using them

- Allow certain users/hosts to be disallowed LAN access and given only Internet access
- Allow certain users/hosts to be allowed only HTTP and have other traffic filtered
- Give users feedback on events that have occurred (access to the LAN denied, etc.)

While these features are certainly not the most compelling that this platform should have to offer, we feel that their implementation will properly guide the development of the platform and the implementations of further components. The successful completion of these tools will then not only serve to provide the capabilities of the tools themselves, but to validate the design and implementation of the framework and the achievement of the high level goals.

Architecture Overview



After analyzing what is needed from our architecture to implement the features outlined above, we have identified the following major components (or classes of components): Evidence Sources, Policy Apps,

the Network View, and the Web Based User Interface (with its two constituent parts: the Front End and the Back End).

The Evidence Sources

Evidence Sources provide information about what users and machines are on the network so that the Network View can accurately provide this information to users and policies. Some evidence sources may be "active", such as one that allows for users to log in explicitly using a "captive portal" type web page. However, we believe the more important sources will be passive and based on the interpretation of network traffic -- sniffing traffic and watching other authentication mechanisms (such as Facebook and email logins) at work and associating those with the high level names (user names) managed by NOX At Home.

Thus, a typical evidence source will register with NOX for packetIn events, sniff the packets it gets for relevant authentication information, and then raise evidence events for which the Network View is listening.

We wish for NOX At Home to adapt to many different usage requirements. To this end, Evidence Sources must be modular units that stand alone as much as possible so that a user can add or remove Evidence Sources to suit their needs. To facilitate that, from the perspective of the architecture, Evidence Sources are just NOX applications that have registered themselves for management by the user interface, and that raise evidence messages. These evidence messages, detailed in Figure 2, may identify either a user or a machine.

One requirement of all Evidence Sources, is that they supply their own user interface (which will be automatically integrated into the larger management UI). This will ensure the greatest amount of flexibility for a framework that must support all possible forms of home networks.

The Network View

The Network View maintains a view of the network state -- which users are on the network, which machines are on the network, which users are on which machines, etc. This is done primarily through the evaluation of evidence given by Evidence Sources by listening to Evidence Messages. We suspect that evidence will seldom be as complete, authoritative, and non-conflicting as we would like, so evidence must be "judged" -- combined by the Network View to arrive at a "best guess" of the truth. This will probably be done using existing research done on the subject of combination of evidence, such as that done by Bayes, Dempster, Shafer, and Hooper.

This "view" of the network is then made available to users (through the Network Browser portion of the UI) and to policies (through an API).

In reference to Figure 4.2, the **Judge** will receive **EvidenceMessages**. It will check *confidenceLevel* as well as *isAutoritative* to understand to validity and weight of the evidence provided. If it is an authoritative source, its confidence will be taken into account for the *validDuration*. Once the **Judge** has made its assessment it will be stored in the **NetworkState**, which can then be queried by the Policies.

The Policies

The Policies are independent NOX applications that, informed by the network state provided by the Network View, determine the behavior of the network. The number of Policy possibilities is endless, however, we will include prototype Policies for HTTP transfer throttling, network guest control, spam protection, and possibly parental controls.

Again the goal is to make the framework as modular as possible. To accomplish this, the removal or addition of a Policy will be as simple as either adding or removing the NOX application followed by the restart of NOX, or, enabling/disabling of the Policy via the Network View's User Interface. An additional complication when compared with Evidence Sources is that Policies may interact or conflict. Thus, a priority system will be included in the Policy design to correct any possible conflicts.

In the same manner as the Evidence Sources, Policies must supply their own user interface. This will ensure the greatest amount of flexibility for a framework that must fit any possible home network behavior desired.

In reference to Figure 4.2, every **Policy** will include its *name*, and will typically provide a *handlePacketIn*which will analyze incoming packets. The Policy will then create the appropriate Flow Table entry with for the network behavior.

The Web Based User Interface

The User Interface will be the key component to make the framework available to all home/SOHO network audiences. We chose to focus on a web-based interface to allow users to access the interface from any machine on the network, requiring nothing to be installed. It is within this User Interface that users and machines will be managed, associations between these users and machines and Policies made, and the selection of Evidence Sources and Policies made. In addition, the configuration of Policies and Evidence Sources through their individual user interfaces will be controlled within this centralized user interface.

The Web Based User Interface itself is composed of two parts: a Front End that runs on the client side (on a user's web browser), and a Back End which bridges the Front End to the rest of our system.

Front End

The front end of the user interface is the portion that users actually interact with -- the portion that is actually loaded into their web browsers. We have chosen to implement this with the qooxdoo. qooxdoo is a framework for the development of rich internet applications using JavaScript. Three major benefits of this approach are:

- Implementation is done entirely using a high-level programming language similar to Java, Python, etc. Developing a polished and sophisticated "web app" using more traditional means (such as direct usage of HTML and CSS) requires a fair degree of specialized knowledge. This would be a burden on our team, but more significantly -- this would be a burden on everyone else who ever wished to extend NOX At Home. Importantly, the people with interest and expertise to implement an new network policy or evidence source are not necessarily skilled at (or interested in) web development. This could easily lead to a situation in which there are interesting pieces of functionality that are completely useless to end users because there is no user interface. A clean and entirely programmatic interface such as qooxdoo provides is similar to those that most "general purpose" developers are already familiar and comfortable with (such as Swing, MFC, Cocoa, etc.).
- qooxdoo has much of what is needed to provide a desktop-like user interface. We foresee that a user may wish to have several pieces of information available at once -- lists of users, lists of user groups, the list of policies, etc. Many web-based user interfaces are divided up into very separate "modes", which are incapable of providing these simultaneously. A standard home WiFi router is a good example of this -- there are usually "tabs" (truly, these are modes) which select between various sections, and a user is limited to working with one section at a time. qooxdoo lets us easily provide a windowed user interface, much like that of a desktop operating system, in which a user is free to open and close any windows (and view and hide any information) that they may wish.
- Modules that must include their own user interfaces can do this simply by providing additional JavaScript code (qooxdoo classes), which can be easily (and programmatically) integrated into the existing UI code, rather than requiring changes to static HTML files.

In reference to Figure 2, the **FrontEnd** will consist of several subcomponents, including **PolicyManager**,**MachineManager**, **UserManager**, **NetworkBrowser**, and **InboxFrontEnd**. Within the **PolicyManager**,**MachineManager**, and **UserManager**, the Administrator will manage the relation of machines to users and how policies will interpret those groups.

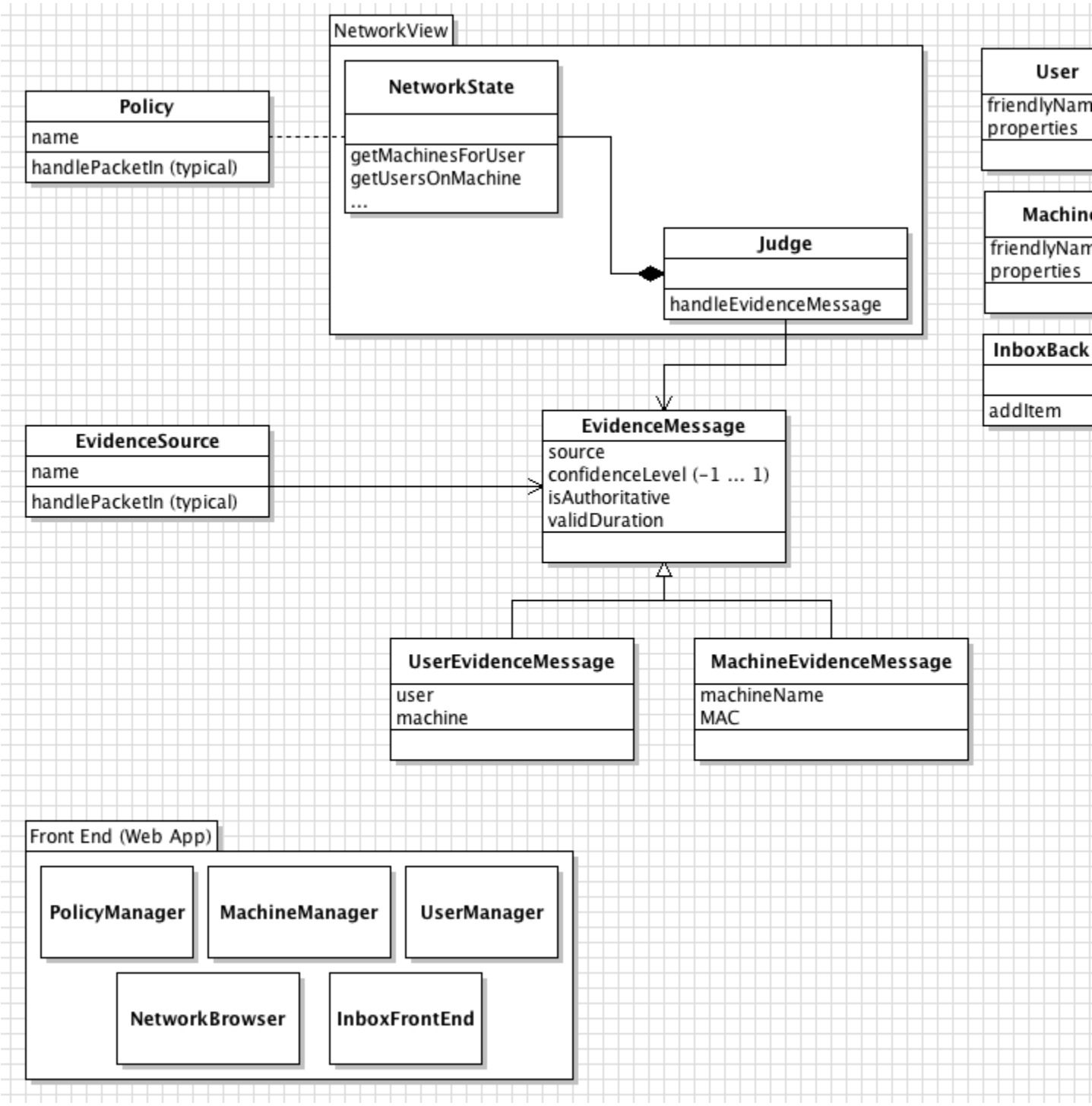
Back End

qooxdoo is an entirely client-side framework that provides the "front end" to our users. Obviously, however, it must pull information from the rest of our software, and allow users to send back configuration changes and the like. This is done through the "back end", which basically just provides a mechanism by which a remote client (running our qooxdoo-based front end) can interface with the rest of our system (such as the Network View and individual Evidence Sources).

This communication will consist of data messages passed through HTTP. The messages themselves will be in JSON (JavaScript Object Notation) format. This is a text-based format derived from JavaScript's object literal notation, but with support in many other languages as well (including Python -- significant for our project).

We hope to utilize NOX's built-in web services mechanism to bridge the gap between JSON-over-HTTP and the rest of our system, though we have not yet made a firm determination that it is a good fit. We may fall back on a third-party solution, such as one based on CherryPy. We intended to investigate this and hopefully make a final determination by our first milestone.

Detailed Module Descriptions



Implementation Plan

In order to build this system, we will be following a modified Lean development process. The process consists of five milestones during which certain functionality is expected to be completed. At the beginning of each milestone, functionality will be divided into chunks and written on cards. Cards will then be assigned to team members. During each milestone the pieces of functionality will be tracked through development phases through the use of a kanban or "visual board". The kanban is broken up into sections correlated with development phases, and the cards representing functionality will be placed and moved within the kanban as appropriate. At the end of each milestone, the development process will be evaluated for efficiency. The process will then be modified per that analysis.

During the initial milestone run, each card will pass through the following three development phases:

Research - In this phase most of the research needed to work on a card is completed, including any major technical hurdles. If, during the course of the research, it is discovered that the original card can not be completed as written it may be broken up, rewritten, or put off until the team can make a decision on it. If the card is sufficiently difficult, a stripped down version of it should be completed, as a proof of concept, before passing on to the development phase.

Development - During this phase the functionality specified on the card should be built. If unexpected difficulty is encountered during this phase the card may be returned to the research phase.

Test - During this phase a developer other than the one who worked on it during the development phase should review the card and make sure it performs as specified. Cards that fail this phase may be passed back to the development phase.

Milestones

The milestones have been selected as "sequence points" where items in a particular milestone would be best served if the items in the previous milestone are complete. Because of the largely agile approach being used, "complete" is a somewhat relative term. At any point, an item may be revisited based on issues discovered in other milestones. However, it would be ideal for the items from previous milestones to be *working* when a new milestone is started.

The dates are not firm deadlines and are mostly meant to ensure that the development scheduling stays on track.

Milestone 1 (March 4th)

- Evidence Apps

    - Machine MAC Address
    - User Email or IM
- Basic Network View with Limited UI

Milestone 2 (March 18th)

- Captive Portal Evidence Source
- Web App
    - Login and Session Management
    - Basic Network Browser UI
    - Basic Inbox

Milestone 3 (April 8th)

- Simple Policies
    - Internet Traffic Only
    - HTTP Only
- Web App
    - UIs for Existing Evidence Apps
    - UIs for User/Machine/Group Management

Milestone 4 (April 22nd)

- UIs for Policy Configuration
- “Other Functionality” UI Framework
- Generic Evidence UIs

Milestone 5 / Completion (May 6th)

- User Testing (will not follow kanban development process)
- UI Review
