Stix: A Goal-Oriented Distributed Management System for Large-Scale Broadband Wireless Access Networks

Giacomo Bernardi
g.bernardi@sms.ed.ac.uk

Matt Calder
matt.calder@ed.ac.uk

Damon Fenacci
d.fenacci@sms.ed.ac.uk

Alex Macmillan
a.macmillan-3@sms.ed.ac.uk

Mahesh K. Marina
mahesh@ed.ac.uk

School of Informatics — The University of Edinburgh, UK

ABSTRACT

`Stix` is a platform managing emerging large-scale broadband wireless access (BWA) networks. It has been developed to make it easy to manage such networks for community deployments and wireless Internet service providers while keeping the network management infrastructure scalable and flexible. `Stix` is based on the notions of goal-oriented and in-network management. With `Stix`, administrators graphically specify network management activities as workflows, which are deployed at a distributed set of agents within the network that cooperate in executing those workflows and storing management information. We implement the `Stix` system on embedded boards and show that the implementation has a low memory footprint. Using real topology and logging data from a large-scale BWA network operator, we show that `Stix` is significantly more scalable (via reduction in management traffic) compared to the commonly employed centralized management approach. Finally we use two case studies to demonstrate the ease with which `Stix` platform can be used for carrying out network reconfiguration and performance management tasks, thereby also showing its potential as a flexible platform to realize self-management mechanisms.

Categories and Subject Descriptors: C.2.3 [Network Operations]: Network Management; C.2.1 [Network Architecture and Design]: Wireless Communication; D.1.7 [Visual Programming]

General Terms: Design, Management, Experimentation, Languages

1. INTRODUCTION

There is a growing recognition of the need for universal broadband across the world with focus on rural/remote areas with low user densities and that on developing countries with poor or non-existent core network infrastructure. It is in this context that *broadband wireless access technologies* (e.g., long distance WiFi [1], WiMax [2]) are seen as a cost-effective alternative to their wired counterparts for bridging the digital divide. The availability of low cost commodity wireless equipment, spectrum regulatory reforms and advances in communications technology have all contributed to this perception and boosted the growth of BWA network operators, usually referred to as Wireless Internet Service Providers (WISPs), around the world. Fig. 1 illustrates a generic broadband wireless access (BWA) network model.

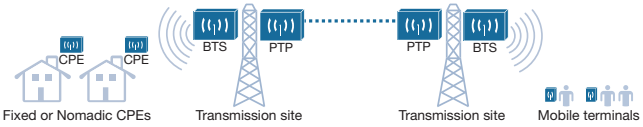


Figure 1: Generic broadband wireless access (BWA) network model, a two tier network: backhaul tier of transmission sites interconnected using long distance point-to-point (PTP) wireless links; access tier providing connectivity to customer premises equipment (CPE) via a point-to-multipoint (PMP) link from the base transceiver station (BTS) located at a nearby transmission site. Note that there could be multiple BTSs at a transmission site, each using a sector antenna. It is also possible that some CPEs may act as relays for other CPEs that are outside the coverage area of a BTS.

Expanding the reach of the Internet to connect the next (few) billion people would mean potentially large-scale BWA networks. Examples include: NGI SpA's BWA network in northern Italy with more than 50,000 subscribers, one of the largest such deployments in Europe [3]; AirJaldi network in northern India with around 10,000 users [4]; and the planned large-scale WiMax roll out in Africa [5].

Network management encompasses a wide range of activities as captured by the ISO FCAPS model, which defines five areas of network management: fault, configuration, accounting, performance and security. Our focus in this paper is mainly performance, and fault and configuration management. These include activities such as monitoring for performance bottlenecks and faults via device level statistics collection, and upgrading the software on managed devices (e.g., PTP devices, BTSs, CPEs). Note that BWA networks (and more generally, wireless networks) are inherently complex to manage [6], given the wide range of parameters and environmental phenomena (e.g., radio propagation, mobility) that can affect network operations.

BWA networks can also be quite diverse from rural/community networks to large-scale WISP deployments, so network management platforms should be adequately flexible to suit a wide range of deployment scenarios. However, *in-band management* (i.e., management traffic going over the production network being managed) is one aspect that is common to all BWA network scenarios as a dedicated management network infrastructure maybe infeasible or impractical due to the additional cost and deployment overhead. Consequently, management traffic contending with user traffic is an issue and as such it is an overhead. This is time of concern with network management systems in use today as they employ a

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
MobiCom'10, September 20–24, 2010, Chicago, Illinois, USA.
Copyright 2010 ACM 978-1-4503-0181-7/10/09 ...$10.00.

centralized management paradigm that puts most of the management related intelligence at one location, typically called a network operations center (NOC). The information about all the managed devices needs to be continually available at the NOC for network monitoring and control decisions. Clearly this becomes an even bigger concern as the network size increases. We will use the term "scalability" to refer to this issue.

Rural or community networks present an additional set of issues: (a) Internet connection is often the most expensive part of the operational expenses for such deployments, so it is not desirable to have communication with the NOC interfering with the efficient use of that precious bandwidth; (b) unreliable connectivity between the NOC and the network being managed seen in rural networks can render the centralized approach ineffective; (c) the number of skilled personnel available to maintain such networks is limited (often, just one person from the community).

Simplifying network management is therefore crucial, more so in developing regions as shown by the experiences by Surana et al. [4]. Simplifying the specification of network management goals is also key for effective distributed management (as outlined under management by objectives paradigm in [7]). Finally, as network size increases, the potential for using heterogeneous devices increases too [4], so it is important to be able to seamlessly support management of devices from different vendors.

In this paper, we develop a novel network management framework and system called `Stix`, motivated by the above considerations. Underlying `Stix` are two key principles: (1) goal-oriented network management; (2) network is the NOC. Regarding the first principle, goal-oriented means allowing the administrator to focus on the network management objectives and the specification of associated activities rather on the low level details. To realize this principle, `Stix` incorporates a high-level visual workflow-based modeling language (referred to as `StixL`) to easily express network management activities.

The second principle is realized by adopting a distributed cooperative agent management architecture for monitoring and control that pushes the manager (called `StixAgent`) of a device being managed closer to that device. A workflow that is designed using `StixL` is converted into an XML file and disseminated to relevant `StixAgents` in the network, which generate executable code on the fly. This principle is essentially aimed at improving scalability by dividing and distributing network management activities within the management infrastructure. Making the management hierarchical and introducing layers of abstraction are well known strategies to tackle network management related scalability and complexity (e.g., arising from heterogeneity) issues [7]. In §3, we elaborate on the types of network management activities that benefit from the `Stix` approach to reduce management related network traffic; briefly, any activity that can be divided and delegated is a prime candidate.

Another unique aspect of `Stix` is that logs generated as a result of executing network management activities (e.g., monitoring statistics) reside *in* the network at `StixAgents` and are replicated locally around the source for high availability using a mechanism called *Sprinkle* that implements a "*log overlay*" within the network. Management data stored on the log overlay is available to the network administrator for on-demand and asynchronous retrieval via a web based graphical user interface with wiki-like syntax and SQL-like querying called `StixView`, thereby decoupling the execution of a network management activity from the retrieval of its results.

`Stix` also provides a *hardware abstraction layer* in the form of a device manager within each `StixAgent` to support devices from multiple vendors. Finally, the `Stix` framework is technology independent, so it can be employed for managing IP-based BWA networks using any underlying wireless communications technology (e.g., outdoor WiFi, WiMax, 4G/LTE).

In summary, the `Stix` distributed management architecture facilitates in-network execution of monitoring and control operations as well as in-network storage of management data. Thus it reduces the reliance on the central NOC for management operations and storage of management data. This, however, does not mean completely eliminating the NOC and the network administrator — they are still needed for specifying network management activities, network visualization/analysis, software updates, billing and accounting operations. Although our context is BWA networks, the `Stix` approach can be applied more widely, e.g., in wired and sensor network settings.

A key contribution of this paper is the implementation of the `Stix` system, including the `StixAgent` on embedded boards. We demonstrate the ability of `Stix` to support management of heterogeneous devices. Using the network topology and logging data from our partner ISP, NGI SpA, we evaluate the scalability and efficiency of the `Stix` distributed monitoring and control approach relative to the traditional centralized SNMP-based management approach and show that `Stix` approach is significantly better. We also present two case studies using the implemented `Stix` system to demonstrate the ease with which it can be used for realistic network management activities such as seamless device reconfiguration and adaptive spectrum management.

In contrast to the commonly used centralized and/or vendor-specific network management tools (e.g., OpenNMS, Nagios, Alvarion Star Management Suite, Motorola Prizm), `Stix` enables distributed and scalable management of large scale multi-vendor BWA networks. Simplifying the specification of network management activities using graphical workflow-based modeling language is a feature unique to `Stix`. Existing work on BWA network management tends to focus on decision making processes in the context of self-management [8, 9] without regard to the underlying implementation platform. No such platform exists currently. *We provide a flexible framework and platform that fills this void, thus enabling self-management of BWA networks*. Although self-management *per se* is not the focus of our work, the case studies as part of our evaluation do demonstrate the feasibility of using `Stix` for self-management processes.

The remainder of this paper is structured as follows. §2 reviews related work. An overview of the `Stix` framework and system is given in §3, and its design and implementation are described in §4. Its efficiency and scalability using data from a real large-scale WISP as well as realistic case studies are presented in §5.

2. RELATED WORK

Pavlou [10] offers a good survey and classification of management approaches that have appeared over the years. Broadly these fall under two categories: management by remote invocation and management by delegation. These two categories roughly correspond to centralized and distributed management approaches, respectively. Remote invocation based approaches can be further divided into manager-agent based (e.g., SNMP, NetConf) or distributed object and service interface based (e.g., CORBA, JRMI, web services). The essential idea behind management by delegation is to move the managing entity closer to the managed device. `Stix` approach falls between the two extremes of management by delegation approach (i.e., manager-agent based and full mobile code based), so can be referred to as a distributed cooperative agent based approach [11].

When it comes to practice, most network management platforms

are centralized (manager-agent based) and based on SNMP (Simple Network Management Protocol), making it the *de facto* standard. While SNMP allows for distributed monitoring with multiple servers (managers) for load balancing and fault tolerance, it does not as such reduce the communication overhead, so it is similar to the default single manager setup from our perspective. `Stix`, on the other hand, deploys managers in the form of `StixAgents` inside the network close to the managed devices (within one hop) with the view of cutting down the communication overhead.

There exists both open source tools (e.g., OpenNMS, Nagios) and commercial tools. In the latter set, many are vendor-specific (e.g., Alvarion Star Management Suite, Motorola Prizm, Ubiquiti AirControl, Meraki's centralized management solution). Though a few multi-vendor tools exist such as AirWave, none of them are for BWA networks. Customizable Wireless Management System (CWMS) [12] proposes to manage heterogeneous BWA networks composed by multivendor devices for both WiFi and WiMax by using metadata expressed in XML to define a glue between different types of devices, but it is a centralized system.

There is limited work on BWA network management [8, 9], mostly focusing on autonomic or self management. These works do not consider the underlying implementation platform for autonomic management in general. This is evident from the lack of usable software implementations from these efforts. Even those that build demonstration prototypes are too naive. For instance, in [8], software running on their access points to implement autonomic processes is in fact statically pre-compiled and pre-deployed C programs, which are inflexible and hard to run in heterogeneous environments; moreover, it does not allow code reuse. In contrast, one of the contributions of our work is to offer a flexible platform to facilitate autonomic management as demonstrated by our case studies in §5. However, autonomic management by itself (e.g., alarm correlation, fault diagnosis, intrusion detection) is not the focus of our work.

As already noted at the outset, the work of Surana et al. [4] is relevant for our work in the sense that it provides a concrete context where simplifying network management and catering to heterogeneous devices are essential. The authors in [4] provide detailed description of the challenges they faced in terms of operational sustainability from their experience with two BWA deployments both having thousands of users. Coming to the network management component, they essentially use a centralized approach via their push-based PhoneHome monitoring mechanism. In contrast to our work, their focus is more on low level issues such as ensuring stable power. There also exists technology specific performance and fault management research for scenarios different to ours, including infrastructure WLANs [13, 14], mesh networks [15] and ad hoc networks [16].

Our workflow-based modeling language (`StixL`) is at a high level similar to the approach adopted in the Click modular router architecture [17]. However, our focus is on device monitoring and control, whereas Click's focus is on packet processing. Also Click's "workflows" are data-bound in that they are triggered on a per-packet or per-frame basis, whereas ours are event-bound and so are fired after a condition (e.g., timer, value of a variable reaching a threshold, message arrival) becomes true. Also the notion of using visual programming languages in general is not new (e.g., OPNET), but our definition and use of a visual programming language (i.e., `StixL`) for goal-oriented network management is.

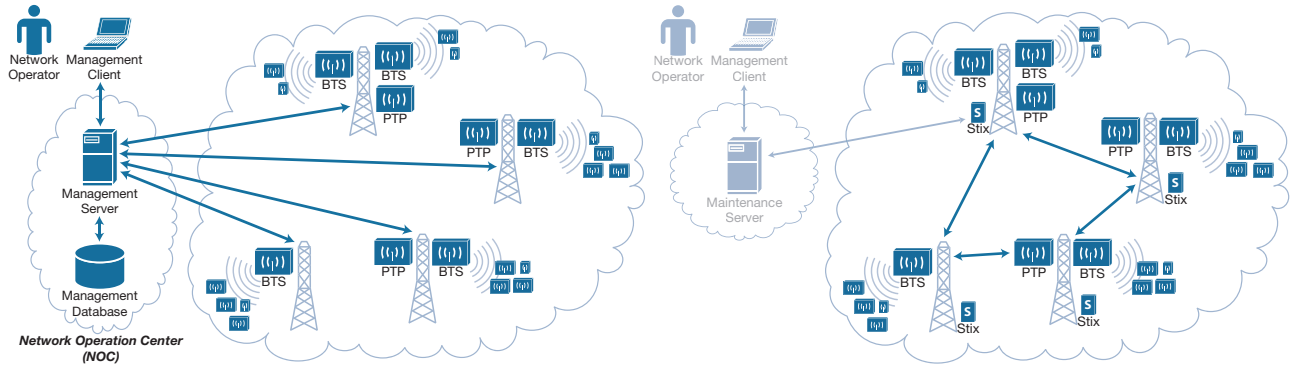
3. STIX OVERVIEW

We consider the problem of developing a network management platform for large-scale broadband wireless access (BWA) networks. As already described, our design requirements for such a platform are as follows: (a) Simplified network management to allow the network administrator focus on management goals rather than burden him with the tedium of low level details. This is especially important in community deployments and developing region settings; (b) Given that management in BWA networks is done in-band, the management platform has to keep the overhead low, thus scale better to large deployments; (c) Reduced dependence on central network operations center (NOC) can potentially let the management system function smoothly even during periods when the NOC is unreachable; (d) It has to seamlessly support multi-vendor devices, which are likely to be the rule rather than the exception in large-scale deployments; (e) Lastly, the platform should be flexible enough to facilitate self management.

We present a novel network management platform for BWA networks called `Stix` that meets the aforementioned requirements. `Stix` design is based on two key principles: (1) goal-oriented management; (2) network is the NOC. Following the first principle, the `Stix` system allows the administrator to describe the goals of the network management activity by modeling processes as workflows, thus meeting requirement (a) above. For this purpose, we introduce the workflow language `StixL`. A workflow is defined as a sequence of tasks that need to be performed in order to achieve a high level network management goal (e.g., upgrade firmware on all CPEs). A workflow can be applied to a specific device or to a set of them with the aid of associated qualifying expressions in a purpose-designed query language; it is formed by combining pre-defined elements such as decision gateways, event triggers and purpose-written code called tasks that takes the form of pluggable boxes to facilitate code reuse. `StixL` also helps realize a distributed management architecture by providing a flexible way for specification of network management activities that are actually executed at management entities inside the network.

`Stix` employs a distributed cooperative agent-based architecture to realize the second principle, thereby meeting requirements (b) and (c). Fig. 2 compares the `Stix` distributed management architecture with the traditional approach. Essentially, to satisfy a network management goal, `Stix` deploys a corresponding workflow to the appropriate set of management entities (referred to as `StixAgents`) situated at transmission sites, which execute the workflow locally and usually upload the results to the *log overlay* using a replication mechanism called *Sprinkle*. Log overlay is in other words an in-network overlay storage system for keeping the logs (e.g., monitoring statistics). Log overlay is asynchronously queried via the `StixView` web interface *as needed* by the administrator to fetch the monitoring results, network health status updates and so forth. Thus the `Stix` system shifts the burden of monitoring, control and storage from the NOC to `StixAgents` via workflows and log overlay store, thus reducing the dependence on the NOC; the NOC is only used for a limited set of operations such as software updates, network visualization, billing and accounting. Hardware abstraction layer within each `StixAgent` helps meet requirement (d), and all the above components in `Stix` collectively satisfy the last requirement (e).

We now discuss the types of network management activities that can lead to reduction in management traffic using `Stix`. It is worth noting that event based management is more scalable and responsive compared to the repeated polling based approach commonly used by network management systems [7]. `Stix` naturally supports event based management as `StixL` workflows are event driven. `Stix` also realizes management by delegation. There are a wide range of network management activities that can be delegated to other management entities or agents in the network such as



(a) Traditional Centralized Management

(b) `Stix` Distributed Management

Figure 2: `Stix` management architecture vis-a-vis traditional centralized architecture.

logging/deduplication/correlation of events, polling of devices for statistics, preprocessing of statistical information and software upgrades; these span all areas of network management from fault and performance management to configuration and accounting management [7]. We consider the specific case of firmware upgrades in our evaluation (§5.1.1). A different way to identify cases benefiting from `Stix` is to look at network management activities as a combination of monitoring and control (especially true for performance and fault management). All such activities that can be done in a self-managing manner as a collection of workflows executing on a distributed set of agents can gain from using `Stix`. Even activities that are inherently centralized and require polling may benefit from in-network processing and aggregation (e.g., usage data for billing purposes). In-network storage and on-demand retrieval of management information further contributes towards system scalability as typically a network administrator is interested in querying a small portion of a network (e.g., a transmission site or a subnet).

In the following, we describe the design and implementation of each component of `Stix`.

4. STIX SYSTEM DESIGN AND IMPLEMENTATION

4.1 StixL: a visual language for describing network processes

From our perspective, a *'network process'* is a set of activities performed during the network lifecycle. Example of processes are routine software upgrades, periodic reporting of link status, emergency routing reconfigurations and so forth. We believe that for a network management scheme to be successful, the administrator must be able to focus only on the operational goals and to express them in a natural language. Our proposal `StixL` is a visual programming language that enables network processes to be described graphically, allows software reuse and hides the complexity of dealing with heterogenous hardware.

`StixL` is loosely derived from the Business Process Modeling Notation (BPMN) [18], a graphical representation developed by the Object Management Group for specifying business processes as workflow. We adapted the concepts from the business community to the specific wireless networking domain. The language is composed of only 17 elements, shown in Fig. 3, which are used in flowchart-like representations called *workflows*.

Round symbols are *events*, and can be classified into three categories: 'start', 'intermediate' and 'end'. Start events trigger the execution of a workflow using timers, incoming messages or conditions. Intermediate events can be used to send and receive messages, to log activities or control the execution flow via error handling. Finally, 'end' and 'terminate' events determine the end of a workflow execution.

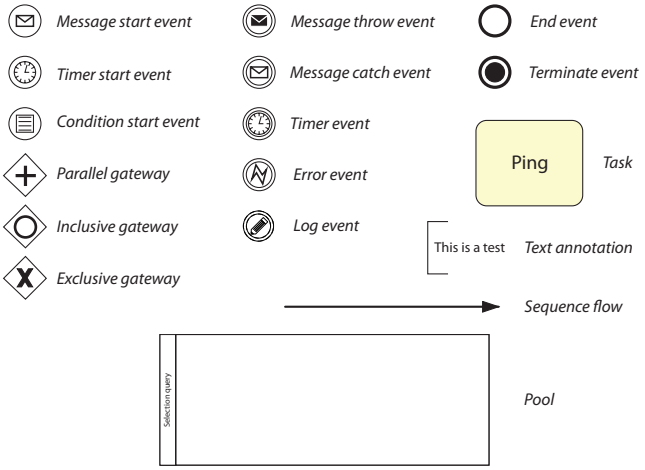
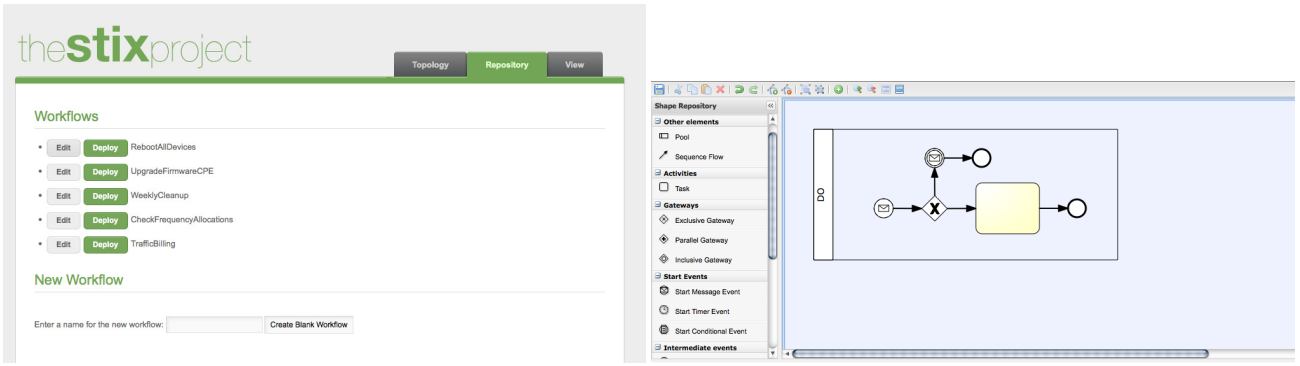


Figure 3: The `StixL` language elements.

A core language concept is represented by the *'task'* element. These are elementary units of work (e.g. a network `ping`, a reboot command, routing control functions) and are implemented as hot-pluggable Java classes. In `Stix`, the administrator is presented a library of common tasks that can be (re-)used in workflows. This library can be continually expanded by writing new tasks or by importing them from 3rd parties.

A workflow is contained in a rectangular *pool* element (Fig. 3). This has an associated query string, which lets the administrator specify the set of devices that fall within the scope of the workflow. The query is expressed in a well-defined syntax, which allows queries to be evaluated against the properties of a particular device. As an example, the query `ON devicemodel='VendorA' AND uptimedays>'10' DO` will cause the associated workflow to apply to all devices of model 'VendorA' with uptime larger than 10 days. We note that while a workflow is executed separately for each device that matches the query executed at the corresponding `StixAgent`[1], a single execution can affect more than one device by using the "message send/receive" events. Additionally, each

[1]In `Stix`, each managed device comes under the purview of a unique `StixAgent`. For example, a CPE is typically managed via the `StixAgent` attached to the transmission site with a BTS that the CPE is associated with.



(a) Repository view

(b) Editor view

Figure 4: Two sample screenshots of the `StixGUI` in action.

workflow has an associated metadata information which includes a globally unique identifier, a revision counter, and optional author names and notes. Metadata also includes option to limit the temporal validity of a workflow (e.g., "don't run before..." and "don't run after...").

`StixL` programs are coded visually using a graphical user interface, which generates an equivalent text serialization in XML format. An XML Schema is used to parse and validate workflows that are exported from the GUI and exchanged between agents.

`StixGUI` is a web-based application that runs on a centralized server and allows the administrator to design and deploy new workflows and edit existing ones. It is divided into three interfaces: a topology view, a repository with the list of existing workflows and the actual workflow editor. The topology view shows the `StixAgents` deployed on the network, with links showing the overall network topology. It is also possible to see the various devices managed at each site (PTP devices, BTSs, CPEs, etc.).

The repository in Fig. 4(a) illustrates all the workflows existing in the network, their name, a brief description, their revision number and other optional details. From this screen it is possible to open the actual process edit window, Fig. 4(b), which we built on the existing Oryx editor project[2]. The editor presents the administrator with the following: a list of `StixL` elements along with the task library; a 'flow mapping' pane to compose the active workflow and wire elements and tasks in sequence; and a 'data mapping' pane to map workflow variables to input and output parameters. These two views are necessary to define both the order in which events happen, also called "execution flow", and to wire the input/output assignment between successive tasks in the workflow (i.e., the "data flow").

4.2 StixAgent: distributed monitoring and control of heterogeneous hardware

`StixAgent` is the core component of the system, as it enables distributed monitoring and control of remote devices. It is internally composed of six parts (Fig. 5):

- a "communication manager" that connects with other agents via SOAP[3] messages. It listens on a network socket for incoming messages and subsequently dispatches them to the appropriate internal part.
- a "workflow manager", which receives new workflows via the communication manager and determines if they are 'relevant' for the locally managed devices. If so, it stores them to disk, transforms the XML representation in appropriate memory instructions and passes them to the workflow engine.
- a "workflow engine", which registers and schedules the execution of a workflow by interpreting its XML representation.
- "log overlay" that keeps track of the most recent log messages originated locally or at the neighboring sites.
- a "storage manager" that provides a persistent storage with an appropriate database interface.
- a "device manager", which is responsible for communication with locally managed devices (see §4.4).

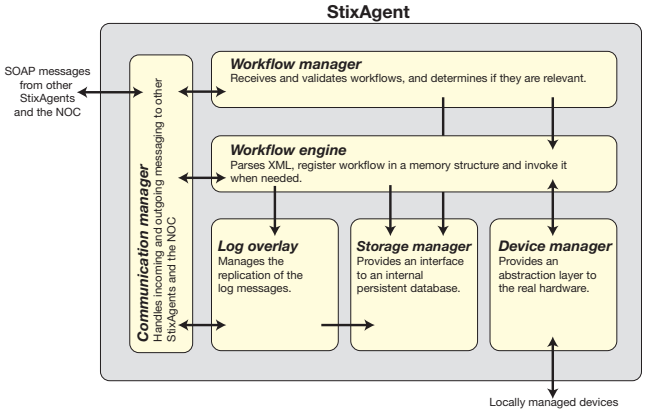


Figure 5: The `StixAgent` architecture.

4.2.1 Communication Manager

`StixAgent` consumes and distributes information by using SOAP messages sent over TCP. Each `StixAgent` runs a server, controlled by the Communication Manager module, which accepts four kinds of XML messages: Log, Event, Workflow and LogQuery. When the Communication Manager receives a message, it determines the message type and finally passes it to the appropriate part within the `StixAgent`.

When workflows are designed and deployed to the network, `StixAgent` forwards them using a purpose-designed flavour of directed flooding, which we call *Pick and Forward* and is implemented between the Communication Manager and the Workflow Manager (described in the next subsection). In this technique, a `StixAgent` forwards a workflow to each of the neighboring `StixAgents` except the one from which the workflow was re-

[2] `http://www.oryx-project.org`

[3] Throughout we use SOAP as the messaging protocol in `Stix` because of its extended support in Java, its lightweight requirements and because it is efficiently verifiable against a schema.

ceived ("Split-horizon"). In case any of the outgoing links is temporarily unavailable, the workflow is locally cached.

4.2.2 Workflow Manager

The Workflow Manager component first performs XML and logical validation on the incoming workflow. For example, the number of starting and ending events are checked from the workflow metadata to assert that there are at least one of each. If validation passes then the Workflow Manager distributes the workflow message to the neighboring agents and also forwards it down to the Workflow Engine. However, in order to preserve memory, not all workflows are registered in memory by the agent. Instead, they evaluate the query string contained in the Pool in order to determine whether the particular workflow will ever be executed on the local devices. Since queries can involve fields that change over time, the pick and forward technique uses simple heuristics to determine whether an agent should pick up and locally store a workflow passing by. For example, a workflow for which the Pool query specifies a minimum value on a monotonically varying field (e.g., the packet counter on an interface) is considered true if the given value is higher than the current field value. By adopting the *Pick and Forward* technique, we are able to save on the memory usage within each `StixAgent`.

4.2.3 Workflow Engine

The Workflow Engine registers the workflow XML in memory, creates a set of supporting data structures and manages the actual workflow execution by interpreting its XML representation. Start events can be triggered by a timer, a condition or an incoming message. In all cases, the engine looks for the corresponding start event, allocates a structure for local variables (which can be optionally declared "persistent"), creates a new thread for the workflow and starts it. The thread interprets the workflow by following the paths through the events and by dynamically loading tasks.

When the execution flows reaches a task, the engine tries to dynamically load the Java class defined for the task. If the binary format of the task is not available locally, the Agent tries to download it directly from the Task Library at the NOC. Each task offers a *runTask* method as an entry point for execution. Once an end or terminate event is reached, the engine deallocates the local variable structure and ends the workflow thread thus freeing its resources. Depending on the execution flow of a workflow (e.g., if there is a loop in the workflow specification), the corresponding thread can be active for an arbitrary long period of time in which case the associated local variable structure remains allocated.

4.2.4 Log Overlay and Storage Manager

Using the 'Log event' element, shown in Fig. 3, a workflow can persistently save any piece of management data. We use the term "log" to refer to such data; logs are represented as tuples containing a timestamp, a reference to the agent that originated it and the data payload. `StixAgents` coordinate to create a 'log overlay' storage in which most recent or important logs generated at an agent are automatically replicated at a few other agents in the network[4]. By doing so, such log data remains available even when the originating site is temporarily down or unreachable and is particularly useful for troubleshooting activities.

For replication, we design a new mechanism called *Sprinkle* that is simple and localized to reduce replication related network traffic. With Sprinkle, a log to be replicated is sent by the originating agent to `jmax` of its `j`-hop neighbors. Each receiving agent will store up to `jnum` messages from the originating agent and will itself pass on the data to `kmax` of its `k`-hop neighbors (with `k > j`). Each recipient of such relayed data will store up to `knum` messages from the originating agent (typically `jnum > knum`). The behavior of the replication mechanism is thus controlled by six configurable parameters. We use `j=1`, `k=2`, `jnum=1000`, `knum=100`, `jmax=2`, `kmax=1` as default values in our implementation. To avoid storing data in poorly connected network regions, each `StixAgent` avoids selection of those neighbors for replication that are solely dependent on itself as a bridge to connect to other transmission sites. Replicating agents determine the log to drop based on a dropping policy whenever the number of logs for an originating agent exceeds the limit (`jnum` or `knum`). In the current implementation, we use a simple policy to maintain only the most recent logs from an agent. It is also possible to extend the log management at replicating nodes to apply an aggregation operation to dynamically reduce the number of remote logs maintained. The mechanism we use for retrieving management information from the log overlay is described in §4.3.

The Storage Manager component is a unified database abstraction, which hides the connection and transaction details of reading and writing to the actual database.

4.2.5 Implementation

For implementation we decided to use Gumstix Overo Earth [19] embedded boards. These feature a System-on-Chip (SoC) based on a 600MHz ARM Cortex-A8 CPU, 256MB RAM and a microSD slot. We added a 4GB memory card and a daughter board, which provides a FastEthernet interface and an I2C bus. The bus is used for `StixControl`, a small low-cost embedded board which we designed to allow power monitoring and control (not discussed here due to space limitations).

The whole setup runs with a single 5V supply and consumes less than 2.5W. The overall cost of a single unit so configured is around 200 USD, but we expect the price to drop significantly for large-scale deployments. Note that one such unit is deployed *per transmission site* to implement the `StixAgent` functionality for all devices managed through that site; this is acceptable when compared to the cost of communication equipment involved in the deployment of a transmission site. We run a modified Linux 2.6.29 distribution and, as Java Virtual Machine, the Sun "J2SE for Embedded 6" [20], which has the advantage of a reduced memory footprint while being largely compatible with the "desktop" J2SE.

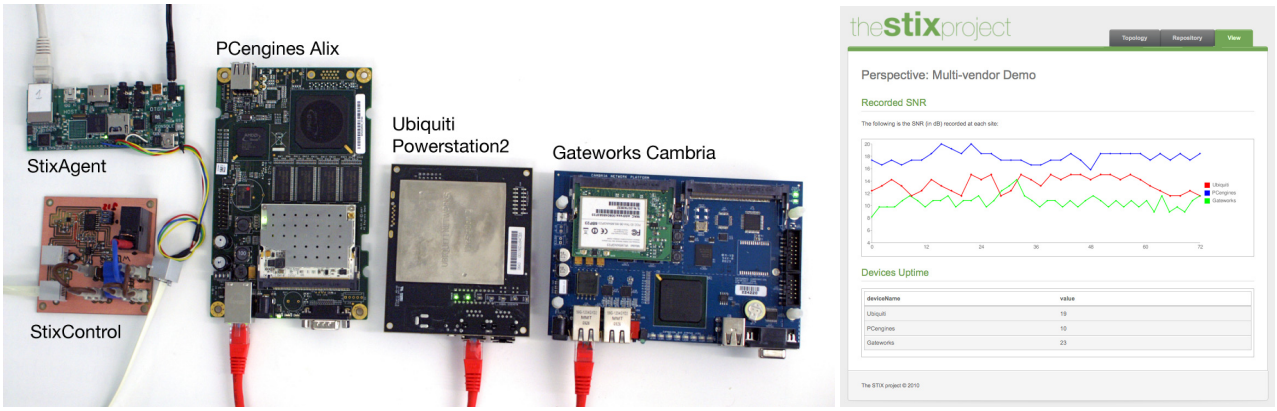
Our implementation of the Storage Manager uses the HSQL database. In practical terms, assuming the average size of a log overlay tuples is 200 bytes, with a 3GB of storage space available in each `StixAgent`, one could store in the local part of the log overlay more than twelve million records, which is reasonably sufficient even for mid to long-term storage.

4.3 StixView: visualizing the distributed network knowledge

`StixView` can be used to generate and view reports, which we call "network perpectives", from the data stored in the log overlay in real time. To enable multiple users to operate concurrently, we modelled perspectives as pages in a wiki engine. A network administrator can create new ones by using a simple syntax that, besides allowing basic text formatting, also includes SQL-like primitives to retrieve and select data from the log overlay. Query results can be rendered in realtime as simple text, as tables or as line, bar or pie charts.

We implemented a two-step rendering engine: in the first step,

[4]Note that all logs generated at an agent are always stored locally regardless of whether or not they are replicated. Locally generated logs in the agent are managed separately from remote logs replicated at the agent.



(a) Sample of devices from our hardware testbed

(b) Screenshot of `StixView` showing data acquired from devices of different vendors

Figure 6: Monitoring of heterogenous hardware using `Stix`.

the perspective page source is compiled into HTML and sent to the web client. Then, Javascript methods scan the document for queries and interrogates the webserver via AJAX calls. A module of the webserver is in charge of interpreting such queries and in turn interrogate the log overlay. Such an approach provides a better user experience — more complex and lengthy queries can be displayed as soon as they are resolved without blocking the rendering of the rest of the page while also allowing multiple queries to be resolved in parallel.

The mechanism for retrieval of management information from the log overlay works as follows: the server always tries to first query the `StixAgent` to which the device in question is mapped to. If that agent is unreachable, then the server queries each of the agent's `j`-hop neighbors, stopping when it receives the requested data. If they are also unreachable, then the same procedure is repeated with each of the `k`-hop neighbors of the original target agent. This on-demand retrieval mechanism results in just one query message in the normal case and O(max. node degree) query messages in the worst case, where max. node degree corresponds to the max. number of neighbors of a transmission site in the network. As a further optimization, the data received in response to prior queries can be cached at the server for fast retrieval of the same data later on. We discuss the impact of different failure patterns in §4.5.

4.4 Hardware Abstraction: the Device Manager

Tasks and events communicate with managed devices through the Device Manager (see Fig. 5), which acts as a hardware abstraction layer. Device drivers implement the interface for each device type and are dynamically loaded as needed depending on the set of devices under consideration. This allows us to flexibly define new drivers to support new devices and to offer a common set of system calls or an API (across all devices) to the workflow engine, while throwing an exception when a system call is not supported by a given device. If the `StixAgent` is running as a software agent, then the Device Manager could just rely on Operating System tools, otherwise via a management protocol (e.g., SNMP).

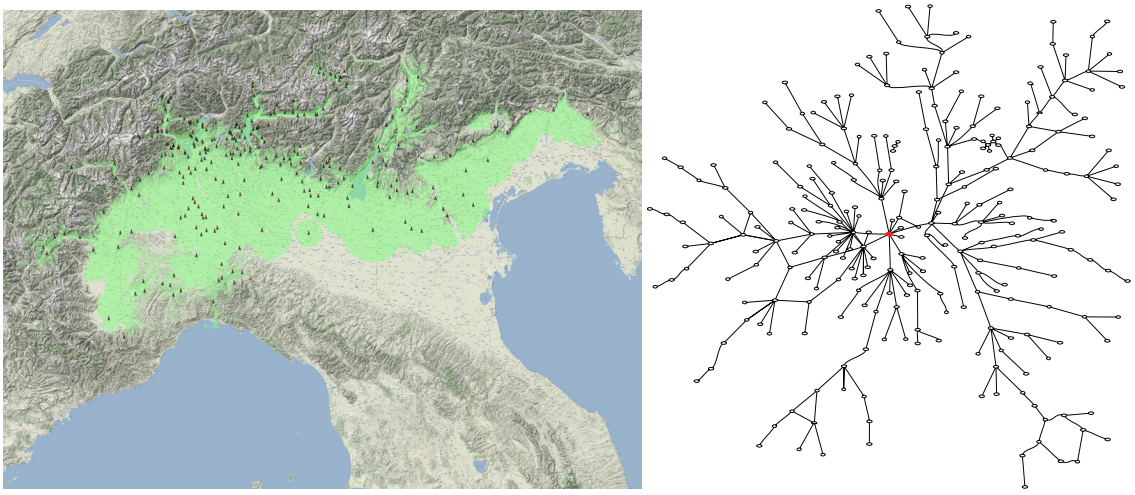
Implementation and Demonstration. Currently, for a hardware based agent, we have implemented drivers for several different devices making use of a handful of protocols (e.g., HTTP, SNMP, SSH).

To demonstrate that `Stix` can support multi-vendor devices, we deployed a small laboratory testbed composed of hardware from different vendors, each equipped with a `StixAgent`, a `StixControl` board. Specifically, we consider different types of wireless devices: PcEngines Alix, Ubiquiti Powerstation2, and Gateworks Cambria (left to right in Fig. 6(a)). Each of these products has a different architecture, and we deliberately configured their software so that they necessitate a diverse spectrum of protocols. The Alix board is managed over SNMP mostly using objects in a private MIB tree, the Cambria device is controlled by issuing commands in a shell over an SSH connection and the Powerstation2 board is managed via its web interface.

The Device Manager in `Stix` hides this heterogeneity via three different device drivers operating over different management protocols, all presenting a common API. The result is that the administrator can design workflows without worrying about the actual type of devices that will be executing them. For example, a 'Reboot' task can be used from the task library and deployed in a workflow,, it will then be up to the device manager to translate the internal `reboot()` method call to the appropriate management procedure. `StixView` is also capable of handling device heterogeneity, for example in Fig. 6(b) we can observe a screenshot of values collected in realtime from the three devices mentioned above.

4.5 Discussion

Robustness to Failures. For `Stix`, we are mainly concerned about transmission site failures as they are the locations where `StixAgents` are deployed. We identify three patterns of site failures: *random* (uncorrelated) failures, where a site becomes unreachable but does not affect connectivity to any other site; *spatially-correlated* failures, in which a set of sites in a specific geographical area is disconnected because of an event in that area (e.g., due to a storm); and *cascade* failures that is a variant of the spatially correlated failures resulting from connectivity failure to a set of sites when a site effectively acting as a "bridge" to the rest of the network fails randomly. The likelihood of such failures is dependent on the redundancy in the network topology. The Sprinkle replication mechanism in `Stix` is inherently robust to random failures. It can be configured to be robust against spatially correlated failures (albeit at a higher replication related communication overhead) when j or k are larger than the typical scope of such failures. Careful selection of replication sites in Sprinkle helps improve robustness to cascade failures. In well-provisioned WISP networks such as the one considered in our evaluation (§5), historical data suggests that site failures are usually rare (as evident from the bottom graph in Fig. 9), and when they do happen they are uncorrelated (mostly due to grid power outage).



(a) Coverage on Map (b) Topology of Point-to-Point Links

Figure 7: Coverage and topology of our partner NGI SpA's BWA network in northern Italy. Each black dot in (a) corresponds to a transmission site. The red dot in (b) around the middle of the graph is the management server node at the NGI SpA's NOC.

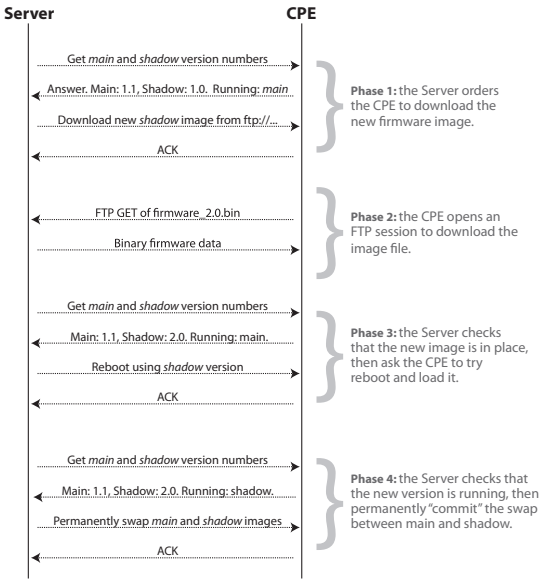


Figure 8: The firmware upgrade dialogue between the centralized management server and a CPE in the NGI SpA network.

Security. While securing access to management schemes is of primary importance, the current `Stix` system does not provide any specific application-level security mechanism, but it relies on an underlying network-level separation (e.g., different VLANs) between user data traffic and management traffic. Segregation of traffic is done routinely on carrier networks to prevent unauthorised access to management interfaces, and can provide further advantages such as prioritization against data traffic. As part of future work, we plan to include a Public Key Infrastructure (PKI) in the `StixAgent` code base. PKI mechanisms would allow developers of `StixL` "tasks" to sign their code, thus enabling the agents to recognize them as trusted. Mandatory access control schemes could also be included to limit users of the GUI to access only to determined region of the network, classes of devices, etc.

To prevent new workflows from causing damage to the network operation, a possible solution is to have a simple runtime simulation scheme: workflow can be simulated and evaluated by adding a "dry run" option in the engine of the `StixAgent` that, while allowing read access to all devices configuration, will "trap" any write access and keep modified variable state in Copy-on-Write (CoW) registers. Our initial investigation shows that implementation of this functionality in the current codebase is possible.

5. EVALUATION

The previous section described our implementation of the `Stix` system and used it to demonstrate the capability for seamless monitoring of multi-vendor devices. In this section, our focus is on evaluating the scalability and efficiency of the `Stix` system, and to demonstrate its utility for BWA network monitoring and control through realistic case studies. We should also mention here that we have just completed augmenting the Tegola network in the northwest of Scotland [21] with a hardware based `StixAgent` at each mast site. We intend to carry out the real-world evaluation of `Stix` using this network.

5.1 Scalability

Here our aim is to quantify the communication and storage overhead of the `Stix` system relative to the traditional centralized management approach for BWA network monitoring and control applications. To do this realistically, we leverage real topology and logging data of our partner NGI SpA [3], which operates one of the largest BWA network deployments in Europe. This network covers most of Northern Italy and its coverage spans more than a third of towns (city councils) in the whole of Italy, covering 3045 from a total of around 8000. As of Feb 2010, NGI SpA's network had 259 transmission sites; 1,112 BTSs and 51,200 CPEs. Fig. 7 shows the coverage on the map and topology showing point-to-point links between transmission sites.

5.1.1 Distributed Control: A Firmware Upgrade Scenario

Nowadays many features of wireless devices are implemented in software, and service providers rely on firmware upgrades to expand the available services and for bug fixes. That makes firmware upgrade a good candidate network control application to quantify

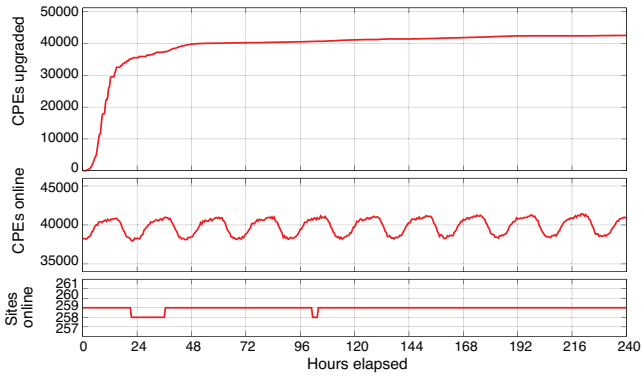


Figure 9: Number of CPEs successfully upgraded in the first ten days (from the time the upgrade was initiated by the server) compared with the number of CPEs and sites online during the same period. Overall, there are 259 transmission sites and 51,200 CPEs in the network. Number of CPEs online shows a diurnal pattern, suggesting that some customers tend to switch off their CPEs at night. Most of the time, almost all transmission sites are reachable.

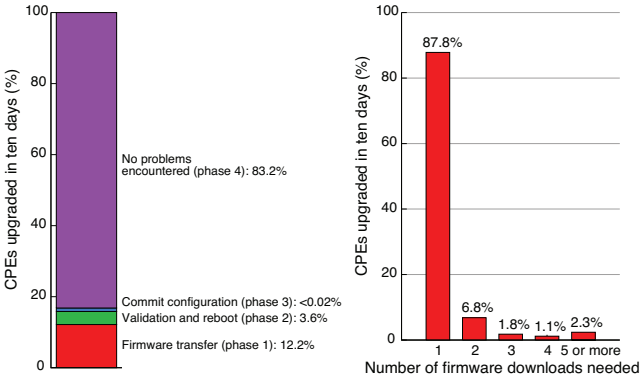


Figure 10: Breakdown of causes behind the failure of firmware upgrade for online CPEs. The graph on the left shows that 16.8% of the online CPEs experience firmware upgrade failure. For 12.2% of the online CPEs, the problem is due to the failure of the firmware transfer; the graph on the right indicates the number of transfer attempts required to resolve this problem for all CPEs.

communication overhead with `Stix` relative to the commonly used approach. CPEs typically being larger in number relative to BTS and PTP devices (also true for the NGI SpA network), we will focus on the case of upgrading the firmware on all CPEs in the network for this scalability assessment. In the NGI SpA network, configuration and management happens over SNMP, and remote firmware upgrade is supported via a commonly used "dual firmware" technique. This technique requires equipping the device with enough persistent memory (i.e., onboard flash) to concurrently store two firmware images. The BIOS then controls which of the two binaries starts at boot based on a configuration parameter. Fig. 8 presents a sequence graph illustrating the communication between the centralized management server and a CPE: the dialogue is composed of four phases that begin with the server asking the CPE to download a firmware image, which is then downloaded by the device being upgraded over a separate session. Then the server asks the CPE to try rebooting using the new image. If the process succeeds, the change is made permanent, i.e., committed.

Clearly the success of such a centralized approach depends on the existence of an end-to-end path between the server and the remote device (CPE in this case). If the CPE is turned off or if there is some kind of communication failure, the process may need to be repeated multiple times. It should be noted that CPEs in the NGI SpA network are outdoor devices installed by qualified personnel on the customer's rooftop, so they should more likely be turned on at all times compared to indoor or mobile CPEs. Nevertheless, it may still take time to upgrade each CPE in the network.

For a recent firmware upgrade performed by NGI SpA in Jan 2010, Fig. 9 shows, from top to bottom, the cumulative number of devices upgraded for the first ten days of the process, the volume of CPEs online in the network and reachable transmission sites at a given time. After upgrades are swiftly performed for a day, the process slows for two main reasons. The first is that the number of CPEs online and still to be upgraded runs out, which cause the centralized server to having to continuously "hunt" for them once they appear on the network. The second is that a remote firmware upgrade can fail even when a CPE is online for several reasons, for which we give a break down in Fig. 10, slowing the process even further.

Now coming to Stix, the network administrator could design a simple workflow that runs on the `StixAgent` at each transmission site *per BTS* such that it gets triggered by the association of a new CPE; at that point it checks whether an upgrade is needed and accordingly performs it. Such a distributed control is beneficial in three ways. The first is that the BTSs have knowledge about the CPEs associated to them, thus they know exactly when to trigger the upgrade operation without needing the central remote server to continuously track for new CPE associations. The second is that the firmware upgrade operation becomes entirely local, making all data-intensive communication (firmware transfer in this case) mostly between the CPE and its BTS; this may greatly reduce the most significant cause of failure we recorded for online CPEs in Fig. 10, i.e., "Firmware transfer: 12.2%". The third is a remarkable reduction in network traffic caused by the firmware upgrade. We estimate this by calculating the total network traffic outgoing from the central management server based on the fact that the firmware image for the CPEs is 944KB in size. This is shown by the shaded area in Fig. 11, which amounts to a total of 40GB in traffic volume. To obtain this plot, we have assumed that firmware transfer for online CPEs succeeds in the first attempt as we do not have access to the detailed time series of failures. Based on the observed failure numbers we obtained (Fig. 10), we can estimate that more than 96,000 firmware transfers had been done in the first 10 days of the process. This roughly corresponds to 94GB of traffic volume,

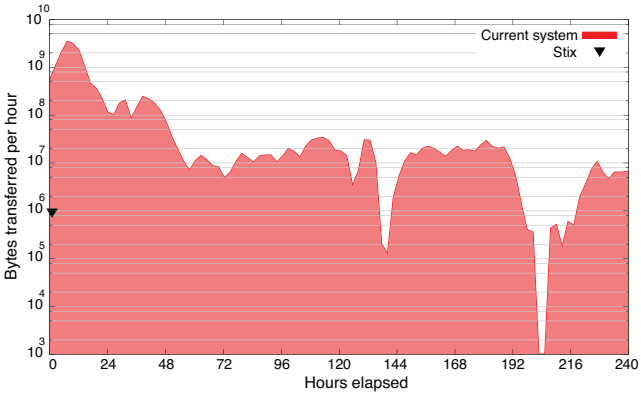


Figure 11: Comparison of estimated network traffic generated for the CPE firmware upgrade application between the current approach used in the NGI SpA network and the `Stix` approach. The shaded area corresponds to a total of 40GB.

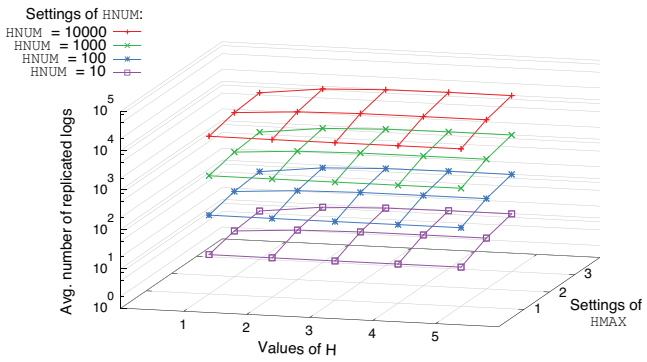


Figure 12: Average number of logs stored at a replicating node on the `Stix` log overlay using the Sprinkle mechanism for the NGI SpA network.

which is more than twice compared to the case with the simplifying assumption in Fig. 11. Using `Stix` instead, only a single transfer of the image to each `StixAgent` is sufficient to delegate the upgrade operation to the individual `StixAgents` at different transmission sites. This results in the total volume of data exiting the central server with `Stix` to be around 1MB, significantly lower than the currently used approach.

5.1.2 *Distributed Monitoring using the Stix Log Overlay*

Here we study Sprinkle's behavior by simulating it on NGI SpA's network topology, shown in Fig. 7(b), to see the impact of the parameters `j`, `k`, `jnum`, `knum`, `jmax` and `kmax`. We use (`h`, `hnum`, `hmax`) as generic variables that could correspond to either (`j`, `jnum`, `jmax`) or (`k`, `knum`, `kmax`). Fig. 12 shows the average number of logs stored at a replicating node on the log overlay as a consequence of a *continuous* stream of logs from an originating agent, calculated on the network of NGI SpA for different settings of `h`, `hnum` and `hmax`. As seen from Fig. 12, the `hmax` parameter can be used to bound the replication overhead in terms of storage for a given set of `h` and `hnum` values. Note that `h` and `hnum` are parameters used for controlling how far and how much to spread the data on the log overlay around the source agent.

Let us consider an example with the following parameter settings to show how the result in Fig. 12 can be used to estimate the storage overhead on the log overlay due to the Sprinkle mechanism for the NGI SpA network.

$$\texttt{j} = 1; \texttt{jnum} = 1000; \texttt{jmax} = 2$$
$$\texttt{k} = 2; \texttt{knum} = 100; \texttt{kmax} = 1$$

This means that, given an originating agent, up to two of its 1-hop neighbors will keep its last 1,000 logs and that up to one of its 2-hop neighbors will keep its last 100 logs. The average number of logs replicated on the log overlay for the agent can be read from the graph as the sum of the $\texttt{hnum} = 1000$ curve at coordinates $(\texttt{h} = 1; \texttt{hmax} = 2)$ with the value of the $\texttt{hnum} = 100$ curve at $(\texttt{h} = 2; \texttt{hmax} = 1)$, which equals 1545. This essentially means that 1000 ($\texttt{jnum}$[5]) logs from the originating agent result in 1545 logs (including duplicates) replicated on the log overlay. In general, `jnum` logs from an agent can produce up to $(\texttt{jnum}*\texttt{jmax}+\texttt{knum}*\texttt{kmax})$ logs on the overlay using Sprinkle. The overhead matches the upper bound for a network topology with node degree greater than or equal to `jmax`. Multiplying by the average size of a log gives the storage overhead due to replication.

[5]Note that `jnum` (> `knum`) is the maximum number of unique logs from the originating agent that can be replicated on the log overlay at a given time.

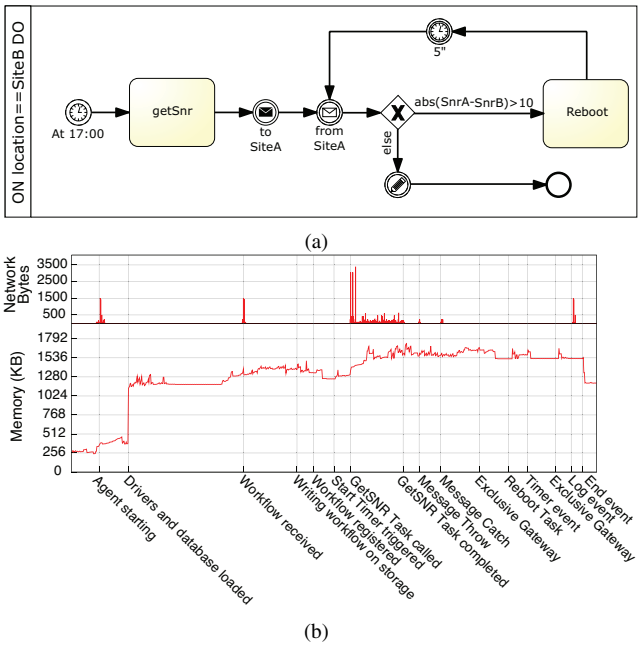


Figure 13: (a) Synthetic workflow used in the resource consumption profiling of the `Stix` implementation; and (b) the resulting network and memory overhead trace over time.

Now turning our attention to replication overhead in terms of communication, with the `Stix` log overlay mechanism, to replicate a log, the communication overhead is upper bounded by the product of $(\texttt{j}*\texttt{jmax}+\texttt{k}*\texttt{kmax})$ and size of the log. This is however reasonable given that `j`, `k`, `jmax` and `kmax` are small constant numbers (1 or 2 in our implementation). In contrast to the above, the communication overhead of the traditional centralized management approach is a function of the path length between the managed device and management server, which can be long (as high as 11 hops in the NGI SpA network).

Retrieval of a log from the log overlay in general depends on the failure probability of transmission sites, Sprinkle parameters and the dropping policy used. Note that replicating nodes are queried only when the original source of the log is down or unreachable. Also, prioritizing certain type of data during replication (e.g., billing related data) would improve its availability in the event of failures. In §4.5, we have discussed Sprinkle's robustness to various failure patterns in general terms. A detailed analysis of the probability of retrieving a log in the presence of failures and limited available storage at replicating nodes requires further investigation. We plan to address this issue as part of future work.

5.2 Efficiency: StixAgent Resource Consumption Profiling

We now evaluate the efficiency of the `StixAgent` implementation in terms of memory and CPU utilization by stress testing it. For this, we created a special synthetic workflow shown in Fig. 13(a) that includes most of the `StixL` language elements: it is started by a timer, then it invokes a Task object that gets the Signal to Noise Ration (SNR) value from a particular wireless interface on the device. That SNR value is sent to a workflow running on another agent, which then answers back with the remote SNR value. An Exclusive Gateway is used to determine whether the difference

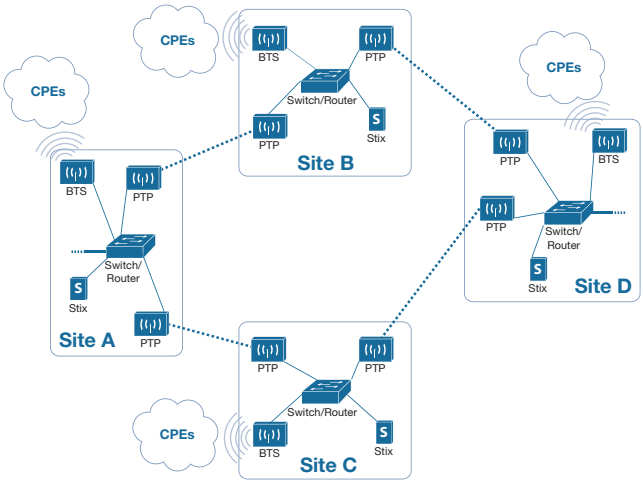


Figure 14: Testbed network used for the case studies.

between the two SNR values is above a threshold. If so, the "Reboot" Task is called, resulting in the local device getting restarted, then a timer waits for the device to come back online. Otherwise, a message is saved in the log overlay and the workflow terminates.

In Fig. 13(b), we show a record of the traffic generated on the network and the memory allocated on the system heap inside the Java Virtual Machine (JVM) at intervals of 10ms. We can observe that the baseline memory footprint of `StixAgent` implementation is around 1200KB without any workflows registered or executing. When the workflow is received from the server and registered, the memory footprint goes up by a modest amount of about 100KB. When it starts executing, this workflow consumes less than 500KB on top of the memory footprint at the time of registration. Although our Workflow Engine implementation within the `StixAgent` has not yet been optimized for memory savings, we believe that memory consumption of our implementation is quite acceptable (compare with the total RAM size of 256MB). This experiment also allows us to estimate the number of concurrent workflows (assuming the one used is a typical workflow) that can run on a `StixAgent` before the OS has to resort to memory paging. The CPU utilization throughout the above experiment remained around 5%, which is again a positive result.

5.3 Case Studies

We now present two realistic case studies to demonstrate the usefulness of the `Stix` system for BWA network management. For these case studies, we created a small indoor lab wireless testbed network as depicted in Fig. 14. We use commodity WiFi hardware for the testbed network. Specifically, each of the PTP links are realized using a different (unused) channel in the 5GHz band, whereas BTS-CPE communication is over 2.4GHz band as in a typical infrastructure wireless LAN. These configurations together allow us to create a BWA network in an indoor setting. A `StixAgent` based on our implementation described in §4 is deployed at each "Site", representing a transmission site in the real world. Routing in this testbed network is performed using OSPF, which also handles link failures.

5.3.1 Seamless Device Reconfiguration

It is increasingly common for ISPs to have "all-wireless" networks in which both the access tier and the backhauling tier are wireless. In these cases, network maintenance operations such as upgrades and reconfigurations are disruptive operation for the customer traffic. As an example, suppose that a major upgrade is to be

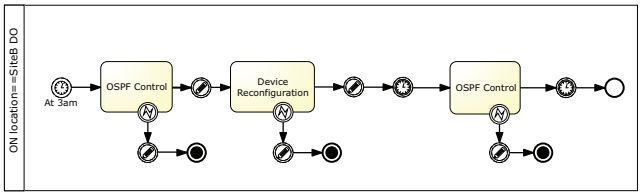


Figure 15: The device reconfiguration workflow.

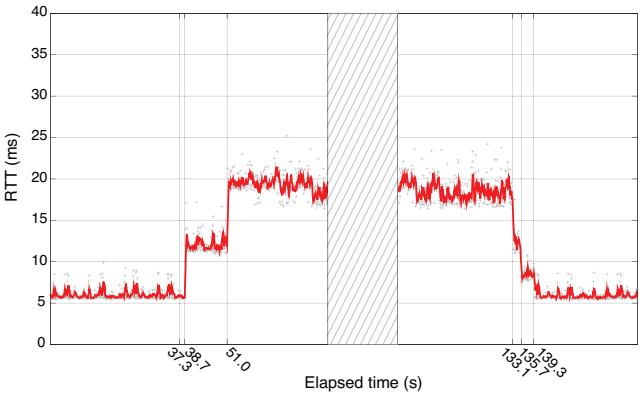


Figure 16: The measured RTT over the network during the execution of the device reconfiguration workflow.

performed at Site B on our small testbed. Such an activity would have impact on ongoing data traffic for the customers connected to the local base station, and potentially for other areas of the network. However, the administrator can design a simple workflow like the one in Fig. 15 to trigger a series of tasks that reconfigure the network routing prior to the upgrade and finally restore the previous routing state. We implemented this by providing an "OSPF Control" object in the `Stix` Task library; it can modify the administrative cost of any local network link based on an input parameter.

The graph in Fig. 16 is a plot of the RTT measured from Site A to Site D of our testbed. Prior to the upgrade, all the traffic is routed via Site B, which offers the best path. When the workflow is triggered, the `StixAgent` running at Site B automatically announces the unavailability of the local links, then waits for the routing modification to be complete, then performs the device reconfiguration and finally announces the availability of the routes. During the time when Site B is not available, traffic is routed via Site C which, despite being a sub-optimal path initially, ensures data packets continue to get routed between Site A and Site D. The step-wise increase in the RTT is because of change in forward and reverse paths at different times.

5.3.2 Adaptive Spectrum Management

Amount of spectrum available for BWA networks is limited, especially in unlicensed bands or those involving nominal license cost (e.g., 5.8GHz band in the UK). This makes adaptive spectrum management a crucial network management activity from a performance standpoint. A simple approach for balancing the number of users (CPEs) associated with a set of base stations (BTSs) has been proposed in [22] for 802.11 networks. In this technique, a central allocation server (which we call *CAS*) determines the spectrum allocation between BTSs based on the number of CPEs at each site, and communicates with each BTS to dynamically adjust the channel width and center frequency. The advantages over static channel width allocation are an increased spectrum utilization and a better per-CPE and per-site fairness.

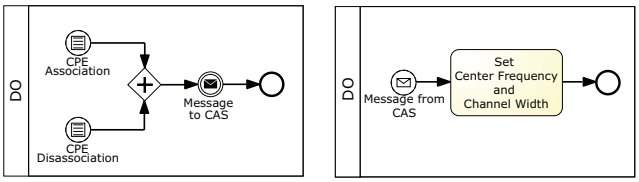


Figure 17: Workflows used for adaptive spectrum management case study.

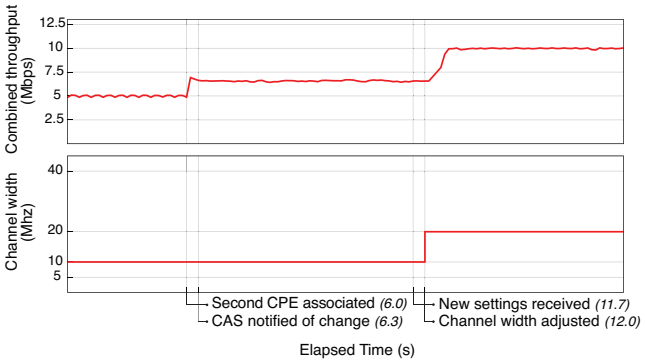


Figure 18: Channel width and aggregate throughput at Site B over time in the adaptive spectrum management case study.

We implemented this mechanism in `Stix` by designing the two workflows shown in Fig. 17, which run on each `StixAgent` for the co-located BTS. The first workflow on the left is triggered by a Condition Start Event every time a new CPE associates or disconnects and causes a message to be sent to the CAS. On the other hand, the second workflow on the right is triggered by a spectrum allocation message sent by the CAS, modifies the center frequency and channel width using the specified Task. In a real world scenario, CAS could be implemented in a server that is external to the network. In our case, it is instead implemented at one of the agents in the network using a workflow (not shown due to lack of space) that is triggered by messages from the workflow shown on the left in Fig. 17.

We deployed the workflows on our testbed network, where a CPE moves from being associated with one BTS to another BTS while receiving a 5Mbps constant bitrate UDP stream from an external server. In this case, all the BTSs operate at the fixed 802.11g datarate of 18Mbps and, initially, the BTS at Site B is allocated a 10MHz wide channel and has only one CPE connected, which is able to receive the stream with 0% packet loss. However, when a second CPE associates to the same BTS, saturation occurs and each of the two streams drops to around 3.5Mbps. As `Stix` reconfigures the network allocating 20MHz to the BTS at Site B, it is able to deliver full 5Mbps again to each of the two associated CPEs (see Fig. 18).

6. CONCLUSIONS

Given the size and complexity of emerging and future BWA deployments, the design of *usable* management systems is hard. We have taken a pragmatic approach to address this challenge. We wanted our system to benefit three categories of users: commercial operators of large-scale BWA networks, non-technical personnel in community deployments and researchers working on self-management techniques. Our contributions can be summarized as follows: (a) `Stix` introduces an easy-to-use visual paradigm to ease the definition of management processes as workflows; (b) it provides a distributed cooperative agent architecture to run such workflowsand store results in the network, thereby reducing management traffic and improving scalability, a fact confirmed by our evaluations; (c) it abstracts hardware heterogeneity and enables "eyes and hands" control to devices being managed. Through our case studies, we have also shown that `Stix` can be useful in easing the implementation of self-management mechanisms. A key part of our future work is to use `Stix` for real-world BWA network management. We plan to do this beginning with our Tegola rural network deployment.

Acknowledgments

We would like to thank our shepherd, Dina Papagiannaki, and Peter Buneman for their helpful comments in revising this paper. We would also like to thank Robert Macgregor for his help in developing `StixControl`, and the Eolo team at NGI SpA for their help and feedback.

7. REFERENCES

[1] L. Subramanian et al. Rethinking Wireless in the Developing World. In *Proc. ACM HotNets*, 2006.
[2] J. Andrews, A. Ghosh, and R. Muhamed. *Fundamentals of WiMAX*. Prentice Hall, 2007.
[3] NGI SpA. http://www.ngi.it/.
[4] S. Surana et al. Beyond Pilots: Keeping Rural Wireless Networks Alive. In *Proc. USENIX NSDI*, 2008.
[5] BBC News. Libya's wireless web access leap. http://news.bbc.co.uk/1/hi/7845905.stm, Jan 2009.
[6] B. Raman and K. Chebrolu. Experiences in using WiFi for Rural Internet in India. *IEEE Communications*, 45(1), Jan 2007.
[7] A. Clemm. *Network Management Fundamentals*. Cisco Press, 2007.
[8] S. Schuetz, K. Zimmermann, G. Nunzi, S. Schmid, and M. Brunner. Autonomic and Decentralized Management of Wireless Access Networks. *IEEE Transactions on Network and Service Management*, 4(2), Sep 2007.
[9] J. Baliosian, K. Matusikova, K. Quinn, and R. Stadler. Policy-based Self-healing for Radio Access Networks. In *Proc. IEEE Network Operations and Management Symposium (NOMS)*, 2008.
[10] G. Pavlou. On the Evolution of Management Approaches, Frameworks and Protocols: A Historical Perspective. *Journal of Network and Systems Management*, 15(4), Dec 2007.
[11] A. Pras et al. Key Research Challenges in Network Management. *IEEE Communications*, 45(10), Oct 2007.
[12] E. Ng et al. Customizable Wireless Management System (CWMS) for Broadband Wireless Access Networks. In *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, 2007.
[13] A. Adya, P. Bahl, R. Chandra, and L. Qiu. Architecture and Techniques for Diagnosing Faults in IEEE 802.11 Infrastructure Networks. In *Proc. ACM MobiCom*, 2004.
[14] Y. Cheng et all. Automating Cross-Layer Diagnosis of Enterprise Wireless Networks. In *Proc. ACM Sigcomm*, 2007.
[15] L. Qiu, P. Bahl, A. Rao, and L. Zhou. Troubleshooting Wireless Mesh Networks. *ACM SIGCOMM Computer Communication Review*, 36(5), Oct 2006.
[16] R. Badonnel, R. State, and O. Festor. Self-Configurable Fault Monitoring in Ad-hoc Networks. *Ad Hoc Networks*, 6(3), May 2008.
[17] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Frans Kaashoek. The Click Modular Router. *ACM Transactions on Computer Systems (TOCS)*, 18(3), Aug 2000.
[18] Business Process Management Initiative. http://www.bpmn.org/.
[19] Gumstix embedded boards. http://gumstix.com/.
[20] SUN J2SE for Embedded. http://java.sun.com/javase/embedded/.
[21] G. Bernardi, P. Buneman, and M. K. Marina. Tegola Tiered Mesh Network Testbed in Rural Scotland. In *Proc. ACM MobiCom Workshop on Wireless Networks and Systems for Developing Regions (WiNS-DR'08)*, Sep 2008.
[22] T. Moscibroda et al. Load-Aware Spectrum Distribution in Wireless LANs. In *Proc. IEEE ICNP*, Oct 2008.