COAP: A Software-Defined Approach for Home WLAN Management through an Open API

Ashish Patro, Suman Banerjee
Department of Computer Sciences, University of Wisconsin Madison
{patro, suman}@cs.wisc.edu

ABSTRACT

In recent years, there has been a rapid growth in the adoption and usage of WiFi enabled networked devices at homes such as laptops, handheld device and wireless entertainment devices. In dense wireless deployments at homes, such as apartment buildings, neighboring home WLANs share the same unlicensed spectrum by deploying consumer-grade access points in their individual homes. In such environments, WiFi networks can suffer from intermittent performance issues such as wireless packet losses, interference from WiFi and non-WiFi sources due to the increasing diversity of devices that share the spectrum. In this paper, we propose a vendor-neutral cloud-based centralized framework called COAP to configure, co-ordinate and manage individual home APs using an open API implemented over the OpenFlow SDN framework.This paper describes the framework and motivates the potential benefits of the framework in home WLANs.

Categories and Subject Descriptors

C.0 [**System Architectures**]; C.2.1 [**Network Architecture and Design**]: Wireless communication

Keywords

Home WiFi networks; Deployment; Routers; COAP

1. INTRODUCTION

Networking at homes continues to get complex over time requiring users to configure and manage them. Central to home networking infrastructure, are wireless Access Points (APs), that allow a plethora of WiFi-capable devices to access Internet-based services, e.g., laptops, handheld devices, game controllers (Wii, XBox), media streaming devices (Apple TV, Google TV, Roku), and many more. Standalone APs today are supplied by a number of diverse vendors — Linksys, Netgear, DLink, Belkin, to name a few. Given its central role in home networks, the performance and experience of users at homes depend centrally on efficient and dynamic configuration of these APs. In this paper, we argue for a *simple vendor-neutral API that should be implemented by commodity home wireless APs to enable a cloud-based management service that enables coordination between neighboring home APs, provides better performance, and reduces burden on users.*

MobiArch'14, September 11, 2014, Maui, Hawaii, USA.

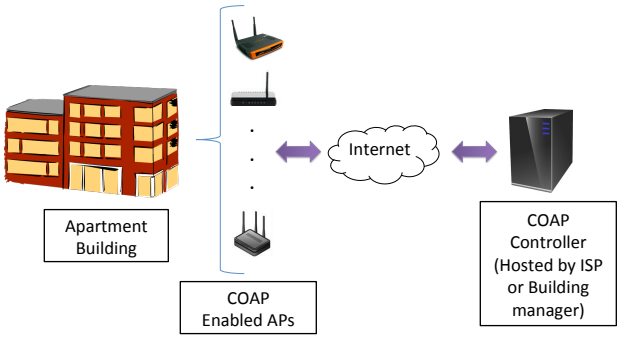

Figure 1: An example of COAP deployment within a residential apartment building consisting of diverse COAP capable home APs and a cloud based controller.

A vendor-neutral API and a cloud-based management service for home wireless APs: In many dense urban environments, a large number of APs (often more than a 50 or 100 of them) from different AP vendors and their associated clients are in range and cause interference to each other. For example, each home AP in our deployment had 20 - 60 neighboring SSIDs. These environments are further challenged by many other wireless devices and appliances, e.g., Bluetooth headsets, cordless handsets, wireless security cameras, and even microwave ovens, that can also operate in the same spectrum and cause further interference. Individual home users neither have the sophistication nor the patience to frequently tune their wireless APs with optimal settings to mitigate the impact of interference.

In our proposed service, called COAP (Coordination framework for Open APs), participating commodity wireless APs (across different vendors) are configured to securely connect to a cloud-based controller (Figure 1). Proprietary cloud-based solutions to manage *individual* home APs [4] exist today. However, such solutions do not attempt to leverage cooperation or coordination between neighboring APs. The controller provides all necessary management services that can be operated by a third-party (potentially distinct from the individual ISPs). In general, it is desirable that all nearby APs use the same controller service for the management function. We believe that our architecture is particularly applicable to large apartment buildings where one could envision that the apartment management contract with a single controller service (e.g., through a fixed annual fee) and all residents are asked to utilize the designated controller service within the building. This service would be no different than many other utilities distributed to residents, e.g., water, electricity, etc., which is arranged by the apartment manage-

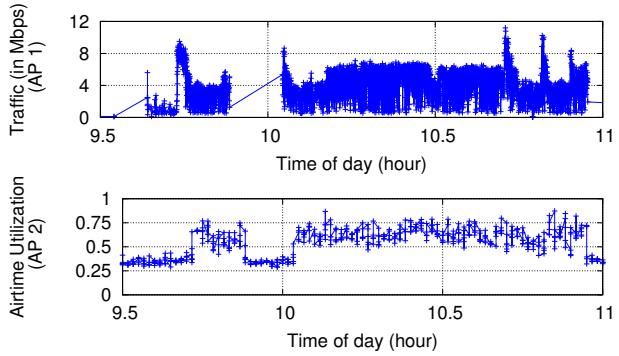

Figure 2: WiFi traffic at AP 1 (top) causing high airtime utilization at a nearby Access Point, AP 2 (bottom).

ment. Individual residents can also pick different controller services to realize many of our proposed benefits. *However, some advanced features, e.g., improved interference management and mitigation, are better served if neighboring APs participate through the same service.*

Using Software-Defined approach in homes vs enterprise wireless networks: The concept of Software Defined Networking (SDN) for WLAN management, using a centralized controller for managing APs is gaining popularity in enterprise WLANs [20]. Further, a few commercial solutions, e.g., Meraki [10], provide vendor-specific proprietary cloud-managed service for APs in enterprise environments. The specific mechanisms used in all such solutions are based on proprietary signaling that primarily manage hardware of a single vendor alone. In this paper, we argue for *an open, vendor neutral API for home APs* that should be supported by each AP manufacturer to allow third-party controller services to be designed, implemented, and deployed for *RF management of home APs*. We believe that a SDN based approach (using an open API) is also important in home environments where each wireless neighborhood has a diverse set of APs. But, unlike enterprise environments, homogeneity in such environments is likely hard to achieve. Furthermore, enterprise WLANs consist of a tightly managed control and data plane with very low latencies between the controller and the APs (order of micro-seconds). It is only possible to create a loosely coupled control and data plane for home settings due to the Internet scale latencies between the APs and controller (tens of milliseconds).

Our proposed API extends the OpenFlow SDN framework [9] and uses the open source Floodlight controller [3] to provide wireless management capabilities. They are complementary to recent proposals [21, 19] that use SDNs to manage residential wired broadband networks. Furthermore, with the growing demands on in-home wireless networks, especially with the dominance and growth in usage of HD media streaming services and devices, the need for coordination between neighbors will continue to grow.

2. WHY USE A CENTRALIZED FRAMEWORK?

In our recent measurement study using a deployment of 30 APs in Madison, Wisconsin [13], we observed many issues due to the ad-hoc deployment of APs in dense residential environments. Following is the summary of the main issues experienced in such deployments.

- **WiFi interference.** Home APs can experience high airtime utilization/channel contention due to factors such as high external traffic and/or use of low PHY rates by legacy clients causing rate anomaly. During periods of active traffic from such transmitters, the average airtime utilization at the neighboring APs increased upto 70%. Figure 2 shows one such scenario observed from the deployment. Traffic from AP 1 resulted in high airtime utilization ($> 70\%$) observed by the neighboring AP 2 operating on the same WiFi channel for more than 1 hour. Such activity from a single AP can increase the congestion experienced by nearby APs. Also, some links can occasionally experience hidden terminal (HT) style interference from nearby APs, resulting in packet losses at the clients and lower throughput.

- **Non-WiFi interference.** Unlike the 802.11 protocol, some non-WiFi devices (e.g., microwave ovens, cordless phones) do not backoff to existing wireless activity on the same channel and can cause packet losses at the receiver due to overlapping transmissions with WiFi packets. In [13], we observed that the microwave ovens caused upto 80% degradation in link performance across multiple locations.

- **Inefficient and static channel assignments.** We also used the deployment in [13] to study the channel assignment patterns of neighboring APs. More than 50% amongst around 300 APs observed during this 1 month period used a single static WiFi channel. This causes the AP to miss multiple opportunities to use better WiFi channels to prevent the aforementioned problems.

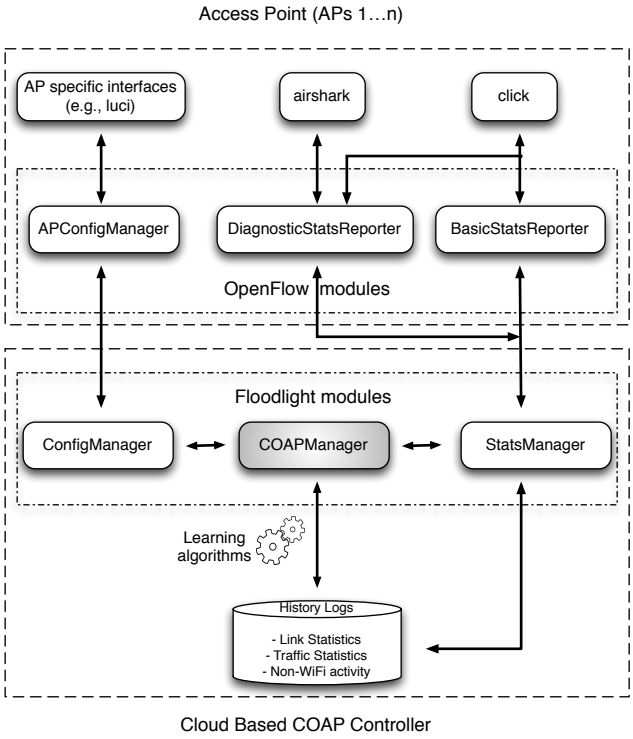

Figure 3: Access Point and controller components of the COAP framework. The "COAPManager" is the main module which manages other modules at the controller.

In each of these cases, if an AP is able to determine the wireless *context* at its neighboring APs and WiFi channels, it can determine the best remedial measure. Individual APs, however, do not have the full view of wireless activities, since each AP can only observe its current channel of operation. If the entire set of nearby APs can

Function	Description
APConfigManager: Module to receive configuration commands from the controller and execute them at the AP	
SetParameters(channel, power)	Configures the AP to a particular channel and/or transmit power.
SetAirtimeAccess(slotDuration, transmitBitmap)	Manage airtime access of APs by throttling/slotting transmissions (§4.2).
BasicStatsReporter: Module to report aggregate wireless statistics to the controller	
GetNeighborInfo()	Scans all WiFi channels for neighboring APs' beacons, gets MAC address (hashed) and RSSI.
GetAirtimeUtilization()	Get the current channel's airtime utilization (0 - 100%) over the most recent time epoch.
GetClientInfo()	Get information about associated clients: e.g., MAC address (hashed), device type.
GetLocalLinkStatistics()	Get packet transmission statistics per local link: e.g., signal strength and total packets sent, received, retried.
GetTrafficInfo()	Get statistics about current traffic: e.g., source id, type, packet count, bytes sent (§4.3).
DiagnosticStatsReporter: Module to report more detailed wireless statistics for diagnosis purposes	
GetNonWifiDevices()	Get neighboring non-WiFi activity (using Airshark [14]): e.g., device type, duration etc.
GetPacketSummaries()	Get fine grained MAC layer packet transmission reports summaries [13] for overheard links. Each packet summary contains: timestamp, packet length, PHY rate, retry bit and RSSI.
GetSyncBeacons()	Get overheard beacon information (receive. timestamp, MAC sequence number, hashed MAC id) on the same WiFi channel for time synchronization (§4.2).

Table 1: Main functions implemented by the COAP APs. Most functions are fairly simple to implement. We are planning to release the API specification and reference implementation: http://research.cs.wisc.edu/wings/projects/coap/.

aggregate this information at a COAP controller service, the latter can then instruct the participating APs on potential adaptations (§4).

3. COAP FRAMEWORK AND API

As discussed in the previous section, some degree of coordination between in-range wireless transmissions is key to achieving improved performance. We now present the details of the COAP framework which uses the OpenFlow [9] SDN framework to centrally manages and enables cooperation between APs across neighboring homes. An important aspect of the framework is that it only requires a software upgrade for commodity home APs and doesn't require any client device support or modifications.

3.1 Framework details

To participate in the COAP framework, APs need to expose a vendor-neutral open API (Table1) to communicate with the COAP controller. COAP is complementary to SDN proposals [21, 19] that manage residential wired broadband networks. To implement COAP, we extend the popular OpenFlow [9] SDN framework to implement the modules shown in figure 3. We implement the COAP controller modules over Floodlight [3] (an open-source Java-based OpenFlow SDN controller) to communicate with the OpenFlow module at the APs. With this setup, our goal is to motivate the usage of SDN style networks to manage residential wireless networks. Other open frameworks (e.g., CAPWAP [5]) can also be used to implement the COAP API.

Our implementation consists of 9000 lines of code (LOC) for the controller, 4000 LOC for the COAP APs and 800 LOC to implement OpenFlow wireless extensions. We have deployed COAP enabled OpenWrt [2] based APs across 12 homes for a period of more than 8 months to gather statistics as well as experiment with the basic management capabilities. Following are additional implementation details about COAP:

Access Points. COAP APs need to implement three different modules that expose a set of capabilities (related to configuration, diagnostics and statistics collection) to the controller. Table 1 describes the different capabilities implemented within each modules. They allow the controller to learn about the wireless activity as well as provides a set of key "hooks" to configure and manage the APs.

In our current implementation for OpenWrt based Access Points, we use the open-source *click* [1] framework to implement the statistics gathering capabilities of the *BasicStatsReporter* and *DiagnosticStatsReporter* modules for the COAP APs. We use it to parse the packet headers of the local and neighboring WiFi links to obtain anonymized aggregate link level statistics. We use Airshark [14] to provide the non-WiFi device detection capability using commodity WiFi cards. The OpenFlow [9] module at the COAP APs communicates with the SDN controller (figure 3) as well as interfaces with Airshark and click-based modules using *netcat* and a standard messaging format. The netcat interface allows AP vendors to replace the click and Airshark modules with their own implementation for reporting WiFi and non-WiFi statistics (e.g., using SNMP) in Table 1.

The *APConfigManager* module interfaces with the OpenWrt configuration tool (*luci*) to perform AP level configurations. (e.g., channel, transmit power). Depending on the AP platform, COAP can interface with other configuration protocols (e.g., netconf [6]). Beyond this basic control, the AP provides an API to remotely manage its airtime access. For example, by slotting time into fine grained intervals (e.g., 10 ms periods), the controller can manage an AP's airtime access to reduce channel contention for important flows and mitigate receiver-side interference (§4.2).

Controller. The COAP controller is implemented over the Java based open source OpenFlow controller, Floodlight, and currently runs on a standard Linux server for our deployment. Floodlight allows developers to use one of the existing modules or implement their own modules within the framework. All COAP controller modules, *StatsManager, COAPManager and ConfigManager*, are implemented as modules within Floodlight. The controller uses the OpenFlow with COAP-specific protocol extensions to collect data from the APs as well as apply the configuration updates.

All server modules and tasks are managed by the *COAPManager* module, through which it manages and configures the connected APs. In the COAP framework, the COAPManager module allows administrators which can allow them to implement custom policies based on their requirements. The server also maintains logs about previous wireless activity (WiFi and non-WiFi) for diagnostic purposes and learn and build models to predict wireless usage characteristics at each AP's location (§4.3).

Wireless extensions for OpenFlow. The OpenFlow communication protocol currently consists of capabilities to exchange switch related statistics (e.g., statistics per switch port, flow, queue, etc.).

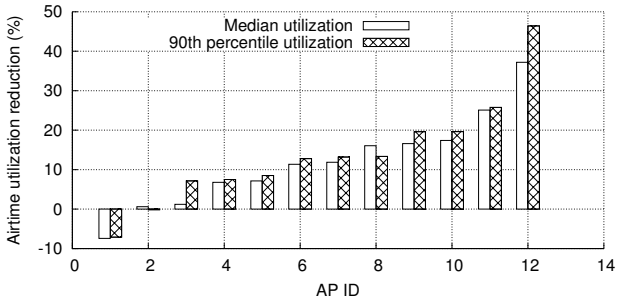

Figure 4: Percentage reduction in median and 90th percentile airtime utilization in our deployment of 12 COAP APs using airtime-based channel configuration.

We augmented this feature to also exchange COAP related wireless statistics (Table 1) and transmit wireless configuration updates.

4. LEVERAGING COOPERATION ACROSS HOME APS THROUGH COAP

By leveraging cooperation between neighboring APs, the COAP framework enables administrators to implement diverse controller applications to enable coordination and improve overall performance. To get maximum benefits from the framework, a single controller should be used for managing COAP APs within a multi-dwelling residential unit (e.g., an apartment building). This is because, coalescing results from multiple APs within the same building at a single controller can allow the COAP controller to create a better wireless performance map for the building. If a single ISP services the entire apartment building, it can host the COAP controller within its premises as another service for its customers. Otherwise, if a building is serviced by multiple ISPs, the building manager can purchase the COAP controller service from third-party cloud based providers.

We now discuss two applications – channel assignment and interference management using COAP airtime management API through a combination of real world deployments as well as and controlled experiments.

4.1 WiFi Channel Configuration

In enterprise WLANs, AP configurations (e.g., channel, transmit power) are managed by a vendor specific central controller. In homes, such configurations are usually performed manually by users or done individually based on local-only observations by APs. Furthermore, channel configurations were observed to be static for more than 50% of home APs [13]. This can lead to inefficient utilization of the spectrum.

In the COAP framework, such functions (e.g., channel assignment) can be delegated to the controller. For example, by combining recent information about airtime utilization on different channels (e.g., within last 'x' minutes) from a COAP AP's neighbors, the controller can perform better channel assignments compared to only local AP observations.

What are the performance improvements due to centralized channel assignments?

To study the benefits of using COAP framework for channel assignments, we performed "airtime-aware" dynamic channel assignments to minimize airtime utilization (and reduce contention) experienced by our residential deployment of 12 COAP APs. Reducing airtime utilization increases the available throughput [16]. Due to the sparsity of our AP deployment, we used a secondary wireless card on our APs to collect airtime utilization information across all channels in a round robin fashion. In this manner, we emulated the usage of nearby APs to collect airtime utilization information.

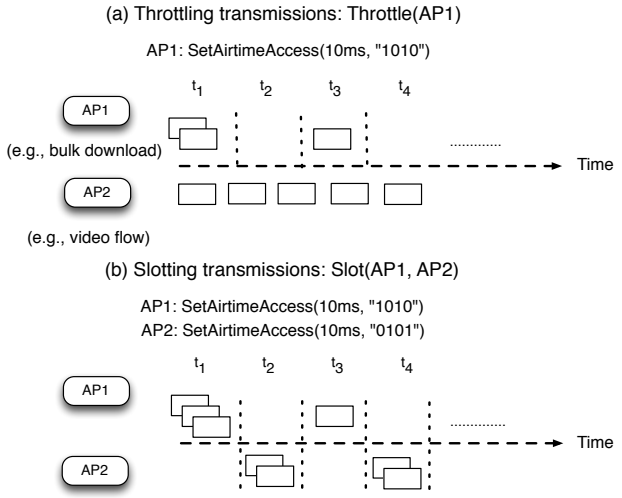

Figure 5: Two mechanisms to manage airtime access of COAP APs: $Throttle(AP_x)$ (top) and $Slot(AP_x, AP_y)$ (bottom). $Throttle$ limits the airtime access of a single COAP AP while $Slot$ coordinates the airtime access of two interfering COAP APs.

Over a span of 6 days, we employed 2 channel configuration strategies on alternate days (1 day per configuration): static (and random) channel assignment, and dynamic channel assignments using COAP to reduce airtime utilization at COAP APs. In the second scenario, airtime utilization statistics from the most recent 5 minute interval was used to estimate external wireless activity and select the best channel per AP [16]. These channel configuration updates were applied during periods without high network activity to avoid service disruption for users.

Figure 4 shows the percentage reduction for median and 90th percentile airtime utilization values across the residential deployments of 12 COAP APs for the COAP-based dynamic channel configuration over the static channel assignment scheme, which is the case across majority of homes today [13] due to lack of centralized management. *It shows that the dynamic "airtime-aware" scheme performed better than a random channel assignment scheme for 10 out of the 12 APs. Four out of the 12 APs (APs 9 - 12) experienced 20% or more (upto 47%) reduction in airtime utilization.* This shows the potential gains achievable through a centralized approach to home AP management. Even non-participating APs in the vicinity of COAP APs can benefit from dynamic channel assignment of COAP APs due to better load balancing of traffic on different channels resulting in lower channel contention. Further proliferation of WiFi-capable devices and high volume traffic (e.g., HD videos) will further increase the future utility of COAP based approaches to reduce channel contention.

4.2 Airtime Management

For the COAP framework, we developed the *SetAirtimeAccess(transmit_bitmap, slot_duration)* API (Table 1) which provides a lightweight and flexible primitive to the COAP controller to manage the airtime access patterns of COAP APs. We propose this API for solving two problems experienced by home APs (§2): (i) Channel congestion caused by nearby AP traffic, (ii) hidden terminal style interference resulting in packet losses. The intuition is to

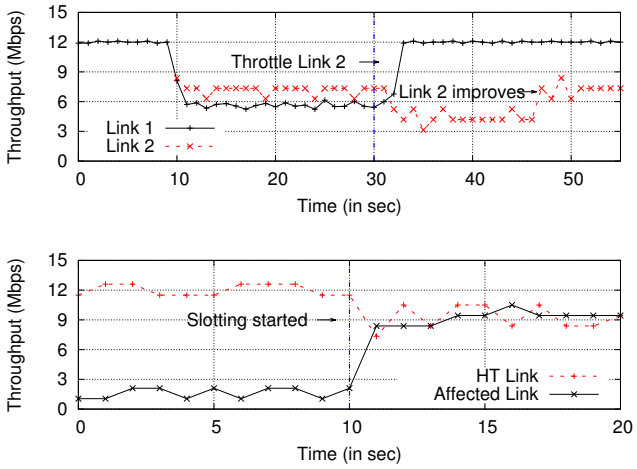

Figure 6: Mitigating interference: (Top) Concurrent traffic from a poor link (Link 2) degrades TCP throughput of link 1; throttling link 2 causes 2x improvement for link 1. (Bottom) Transmissions from a hidden terminal degrades a nearby link's performance; slotting the two APs improves the affected link's throughput by $4\times$.

mitigate these scenarios by controlling the airtime access patterns of the interfering APs.

For example, the *impact of hidden terminal interference can be mitigated by preventing overlapping transmissions between the two interfering APs*. To detect hidden terminal interference, we borrow techniques from prior work [18, 13] that correlates time-series packet losses and activity of nearby links. Similarly, *channel congestion experienced by high priority flows (e.g., VOIP, HTTP based video flows) can be mitigated by reducing the airtime access of nearby APs with low priority flows* (e.g., bulk downloads). Lightweight and coarse-grained classification techniques (e.g., [11]) can be used to detect flow types and identify these sensitive flows.

4.2.1 API details

We use two mechanisms to manage the airtime access of COAP APs (Figure 5): $Throttle(AP_x)$ and $Slot(AP_x, AP_y)$. For this purpose, the controller divides time into small "slots" (10 - 20 ms) and can enable/disable transmissions of COAP APs on a *per-slot* basis to manage their airtime access. $Throttle(AP_x)$ is used to limit a COAP AP's transmission to certain slots to decrease its airtime utilization, and thus reduce channel contention at the neighboring APs. $Slot(AP_x, AP_y)$ is used by the COAP controller to configure two APs to transmit in *non-overlapping time slots*. This is useful in mitigating hidden terminal interference since overlapping transmissions cause packet losses at clients.

In contrast to enterprise-centric centralized approaches such as CENTAUR [17] that explicitly schedule flows on the data path to mitigate interference, it is not possible to do across home APs due to the lack of a centralized data-plane. But the centralized SDN framework still allows the COAP controller to coordinate transmissions between nearby home APs using the aforementioned API. This approach is motivated by RxIP [8], which uses a distributed mechanism for coordinating AP transmissions but can result in scalability issues in large multi-dwelling residential units.

API usage. To use this API, COAP APs are configured with the two parameters: *"slot duration"* and *"transmit bitmap"*. The *slot duration* (e.g., 10 ms) determines the time granularity to use for airtime access management (enable/disable transmissions). The per-AP *transmit bitmap* specifies the slots in which the AP is allowed to transmit. For example, a transmit bitmap of "1010" indicates that the AP is allowed to transmit in time slots 0 and 2 and disabled in slots 1 and 3 (Figure 5, top). This pattern represents a throttle value of 50% since the AP can only transmit data packets during 50% of the time.

For the $Slot(AP_x, AP_y)$ mechanism, the COAP framework uses a passive mechanism from [15] to synchronize the APs' clocks with < 0.5 ms error which is much smaller than the slot duration of 10 to 20 ms.

Implementation. We have instrumented the ath9k wireless drivers for our 802.11n based APs to support this API. For example, to throttle the airtime access of an AP, we disable the AP's transmit queue (using the AR_Q_TXD register) to block packet transmissions for the required period of time. Since this is a software-only modification, the underlying implementation of this feature can be driver specific and transparent to the COAP API. We explain the now usage of this API using a couple of example scenarios.

Example Scenarios. We now present a couple of example scenarios to show how the COAP controller can manage the links' airtime access using the $Throttle(AP_x)$ and $Slot(AP_x, AP_y)$ mechanisms.

In the first scenario (Figure 6, top), the iperf throughput of Link 1 at 12 Mbps degrades due to overlapping transmissions from a poor link (Link 2) which uses low PHY rates (at t = 10 seconds). At t = 30 seconds, the controller throttles Link 2 by limiting its transmissions to alternate 10 ms time slots (50% throttle), This improves the performance of link 1 from 6 Mbps to 12 Mbps due to the sufficient airtime available to get the desired throughput. At t = 45 seconds, the link quality of link 2 improves, allowing to get higher throughput (from 4Mbps to 7.8 Mbps) with the same 50% airtime throttle. Thus, this API provides a strict control over an AP's transmissions which is *independent of the varying link + PHY layer properties (e.g., signal strength and data rates)* and provides protection to important flows.

In the second scenario (Figure 6, bottom), a link experiences poor throughput (2 Mbps) due to interference caused by a nearby hidden terminal AP resulting in high losses and backoffs. At t = 10 seconds, the controller detects this interference and protects the affected flow by configuring the two APs to transmit in alternating and non-overlapping time slots (using transmit bitmaps of *"1010"* and *"0101"*). The protection provided by slotting increases the throughput of the affected link to 9 - 10 Mbps ($4\times$ improvement) while the hidden terminal AP still obtains a throughput of 8 - 10 Mbps.

4.3 Context-aware AP Configuration

For other potential applications, the controller can utilize server logs about previous wireless activity observed by each AP to apply learning-techniques to mine context-related information. One example is the "time-of-day" context about previous non-WiFi activity at a particular location. If the usage of some non-WiFi device (e.g., cordless phones, microwave ovens) is highly correlated with a particular period of the day, the controller can pre-emptively configure the APs (e.g., channel) to avoid interference from the particular device.

Another example is the "client type" context to predict future wireless activity. For example, if a WiFi client associated to an AP (e.g., Wireless TV) has long periods of continuous activity, the controller can leverage this information for future configuration of the AP and its neighbors. Following is an example scenario – *"AP 4 has*

started a high-volume HD video flow on his wireless TV, which is going to last for 30 minutes with 90% probability. Therefore, move adjacent APs to other WiFi channels to reduce channel contention during this traffic's presence."

5. RELATED WORK

SDN style management of APs. Meraki [10] pioneered the concept of cloud-based management of enterprise APs based on an proprietary solution that configures their homogeneous APs remotely though the cloud. Dyson [12] proposes a centralized framework to manage APs and clients in enterprise WLANs using a set of APIs at both APs and clients. We propose an open API by augmenting existing SDN frameworks to enable cooperation between heterogeneous neighboring APs in residential settings. OpenFlow [9] is an open and popular SDN framework designed to manage routing policies and other networking-related configurations. Prior work on using OpenFlow for 802.11 networks [7, 20] focused on building an experimental platforms for enterprise network deployments. Our goal is to use the SDN approach to actively cloud-manage wireless parameters of 802.11 networks and build management frameworks, especially for residential settings. It is also possible to use other open AP management standards (e.g., CAPWAP [5]) to implement COAP related APIs to manage residential AP deployments.

Using multi-AP support in non-enterprise 802.11 deployments. Prior research has proposed mechanisms to leverage support from multiple APs to use AP virtualization to improve video performance through bandwidth sharing [19] and mitigate hidden terminals using distributed algorithms [8]. With COAP, we motivate the use of a cloud-based framework to enable cooperation between nearby home APs and mitigate wireless problems in residential WiFi deployments. This is done by aggregating airtime utilization information from nearby APs for channel configuration, coordinating airtime access across APs to alleviate both channel congestion and hidden terminal scenarios and learning from prior activity to predict traffic usage patterns and future non-WiFi interference.

6. CONCLUSION

We presented the COAP framework, which uses a vendor-neutral open API and a cloud based controller to enable cooperation between heterogeneous and co-located home WiFi APs in dense residential deployments (e.g., apartment buildings). We describe multiple applications to motivate the benefits of using this centralized framework in such deployments – better channel assignments using airtime utilization information from co-located home APs and managing airtime access of neighboring APs to reduce channel contention for important flows as well as mitigate hidden terminal interference. In future work, we plan to explore the possibilities of leveraging the COAP framework to develop context-aware applications and apply pre-emptive configurations at APs to mitigate performance issues.

We will continue to refine our controller implementation as well as our API specifications as we continue to learn from our deployments. We also plan to open-source our COAP implementation in near-future to motivate and engage developers, vendors and other stakeholders to encourage the adoption of this framework. We'll be publishing these updates at: http://research.cs.wisc.edu/wings/projects/coap/.

7. ACKNOWLEDGMENTS

This work is supported in part by the US National Science Foundation through awards CNS-1040648, CNS-0916955, CNS-0855201, CNS-0747177, CNS-1064944, CNS-1059306, CNS-1345293, CNS-1343363, CNS-1258290, and CNS-1405667 and by a Microsoft Research PhD Fellowship.

8. REFERENCES

[1] Click modular router. http://www.read.cs.ucla.edu/click/click.

[2] Openwrt. https://openwrt.org/.

[3] Project floodlight. http://www.projectfloodlight.org/floodlight/.

[4] D-Link. Cloud router. http://www.dlink-cloud.com/solutions.aspx.

[5] IETF. Capwap protocol specification. http://tools.ietf.org/search/rfc5415.

[6] IETF. netconf. http://datatracker.ietf.org/wg/netconf/charter/.

[7] Kok-Kiong Yap et al. The Stanford OpenRoads Deployment. In *Proceedings of the annual conference on WinTech, 2009*.

[8] J. Manweiler, P. Franklin, and R. Choudhury. RxIP: Monitoring the health of home wireless networks. In *INFOCOM 2012*.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 2008.

[10] Meraki. Enterprise cloud management. http://www.meraki.com/products/wireless/enterprise-cloud-management.

[11] A. W. Moore and K. Papagiannaki. Toward the accurate identification of network applications. PAM'05.

[12] R. Murty, J. Padhye, A. Wolman, and M. Welsh. Dyson: An Architecture for Extensible Wireless LANs. USENIX ATC'10.

[13] A. Patro, S. Govindan, and S. Banerjee. Observing Home Wireless Experience Through WiFi APs. MobiCom '13.

[14] S. Rayanchu, A. Patro, and S. Banerjee. Airshark: Detecting non-WiFi RF devices using commodity WiFi hardware. IMC '11.

[15] S. Rayanchu, A. Patro, and S. Banerjee. Catching Whales and Minnows Using WiFiNet: Deconstructing non-WiFi Interference Using WiFi Hardware. NSDI'12.

[16] E. Rozner, Y. Mehta, A. Akella, and L. Qiu. Traffic-Aware Channel Assignment in Enterprise Wireless LANs. In *ICNP 2007*.

[17] V. Shrivastava, N. Ahmed, S. Rayanchu, S. Banerjee, S. Keshav, K. Papagiannaki, and A. Mishra. CENTAUR: Realizing the Full Potential of Centralized Wlans Through a Hybrid Data Path. MobiCom '09.

[18] V. Shrivastava, S. Rayanchu, S. Banerjee, and K. Papagiannaki. PIE in the sky: online passive interference estimation for enterprise WLANs. NSDI'11.

[19] V. Sivaraman, T. Moors, H. Habibi Gharakheili, D. Ong, J. Matthews, and C. Russell. Virtualizing the Access Network via Open APIs. CoNEXT '13.

[20] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao. Towards programmable enterprise WLANs with Odin. HotSDN '12.

[21] Y. Yiakoumis, K.-K. Yap, S. Katti, G. Parulkar, and N. McKeown. Slicing home networks. In *Proceedings of the 2nd ACM SIGCOMM workshop on Home networks*, HomeNets '11.