Towards Programmable Enterprise WLANs with Odin

Lalith Suresh
Instituto Superior Tecnico
Lisbon, Portugal
lalith.puthalath@ist.utl.pt

Julius Schulz-Zander
Telekom Innovation
Laboratories / TU Berlin
Berlin, Germany
julius@
net.t-labs.tu-berlin.de

Ruben Merz
Telekom Innovation
Laboratories / TU Berlin
Berlin, Germany
ruben.merz@telekom.de

Anja Feldmann
Telekom Innovation
Laboratories / TU Berlin
Berlin, Germany
anja@
net.t-labs.tu-berlin.de

Teresa Vazao
INESC-ID / Instituto Superior
Tecnico
Lisbon, Portugal
teresa.vazao@ist.utl.pt

ABSTRACT

We present Odin, an SDN framework to introduce programmability in enterprise wireless local area networks (WLANs). Enterprise WLANs need to support a wide range of services and functionalities. This includes authentication, authorization and accounting, policy, mobility and interference management, and load balancing. WLANs also exhibit unique challenges. In particular, access point (AP) association decisions are not made by the infrastructure, but by clients. In addition, the association state machine combined with the broadcast nature of the wireless medium requires keeping track of a large amount of state changes. To this end, Odin builds on a light virtual AP abstraction that greatly simplifies client management. Odin does not require any client side modifications and its design supports WPA2 Enterprise. With Odin, a network operator can implement enterprise WLAN services as network applications. A prototype implementation demonstrates Odin's feasibility.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network management; C.2.1 [**Network Architecture and Design**]: Wireless Communication

Keywords

Odin, SDN, Enterprise WLANs, Virtualization, OpenFlow

1. INTRODUCTION

Deployments of modern IEEE 802.11 [1] enterprise wireless local area networks (WLANs) can range from a few dozen to thousands of access points (AP), serving a multitude of client devices such as smart-phones, laptops or tablets. Large deployments provide resilience, fault-tolerance and fail-over capabilities. Scalability in this context is paramount. Independent of the size, what most enterprise WLAN deployments have in common is the support for features such as authentication, authorization and accounting (AAA), policy management, mobility management, interference management with dynamic channel reconfigurations or client re-associations, load balancing and providing quality of service guarantees. Their management is usually centralized. These systems are also proprietary, with each vendor offering its own closed-source platform.

In this paper, we propose Odin[1], a prototype software defined networking (SDN) framework for enterprise WLANs. The objective of Odin is to empower network operators to program and deploy typical enterprise WLAN services and features as network applications. In this context, WLANs exhibit specific challenges. The 802.11 standard lets clients make AP association decisions on the basis of locally made decisions. The infrastructure has no control over these decisions made by the client. In addition, the association state machine at the AP, combined with the dynamic, broadcast and time-varying nature of the wireless medium can require to keep track of state information changes continuously. Furthermore, not only associated, but also all interfering 802.11 devices need to be managed.

To simplify the programming model, for instance, to not have to handle callbacks for 802.11 state machine changes, Odin builds on a light virtual access point (LVAP) abstraction. LVAPs virtualize association state and separate them from the physical AP. For the programmer, multiple clients connected to a single physical AP are treated as a set of logically isolated clients connected to different ports of a switch. The LVAP design not only prevents from directly handling the association state, it also greatly facilitates mobility management. It allows for the infrastructure to handoff clients without triggering the re-association mechanism.

Odin is work in progress, under heavy development and currently makes several assumptions: It targets fully centralized and reasonably sized deployments with one controller.

[1]Odin is a major god in Norse mythology.

It does not expect any client side modifications and its design supports WPA2 Enterprise. Clients connect to an Odin network as regular IEEE 802.11 stations in infrastructure mode. Odin is fully transparent to the client.

There is much literature available on enterprise WLAN architectures. Systems like CENTAUR [2] and Dyson [3] propose architectures that assume heavy modifications to the IEEE 802.11 [1] client side logic. Systems like DenseAP [4] and DIRAC [5] assume legacy 802.11 clients. However, these systems have certain performance limitations. For instance, they cannot work around the delays involved with a handoff by a client. With regards to improving performance in the network, Rozner et al. describe traffic aware channel assignment in enterprise WLAN deployments [6], and Bahl et al. [7] propose the use of desktop computers as monitoring stations in order to collect statistics about the network, which can be used for making network management decisions. For Odin, these works represent typical applications that we expect to eventually support. Odin is built on top of an OpenFlow controller [8]. Previous work on OpenFlow for an 802.11 and WiMAX environment [9] concentrates on supporting concurrently running experiments. Odin goes much further by recognizing and taking into account peculiarities of the 802.11 environment. Recently, research efforts like Frenetic [10] and NetCore [11] were made in the direction of programming languages and compilers for network programming. These efforts are complementary to the Odin framework. Finally, Odin also borrows ideas from Onix [12], which treats a network as an eventually consistent graph with different elements. With Odin, we seek the high-level primitives that a programmer needs within the context of enterprise WLANs.

The remainder of this paper is organized as follows. The design of Odin is presented in Section 2. A set of Odin applications are described in Section 3. Section 4 discuss implementation details of Odin. The feasibility of Odin is demonstrated in Section 5 in a small scale deployment. Finally, we detail current work and conclude the paper in Section 6.

2. ODIN FRAMEWORK DESIGN

To effectively express applications that implement high-level services in enterprise WLANs, the programmer needs simple and powerful abstractions. This is essential if the programmer operates on a central view of the entire network. However, 802.11 clients associate or re-associate with any AP on the basis of locally-made decisions. A design goal of Odin is to prevent the programmer from keeping track of changes to the authorizer, authenticator and client state machines. Indeed, the programmer cannot make assumptions about the endpoints of the link between the client and the infrastructure (MAC and IP address). To shield the programmer from this burden and to simplify the programming model, we propose light virtual access points (LVAP). With LVAPs, programmers always see a fixed link between clients and the infrastructure (Section 2.2). We first explain Odin's architecture.

2.1 Architecture of Odin

Odin's architecture consists of a single master, multiple agents and a set of applications (Figure 1). The master itself is an application on top of an OpenFlow [8] controller. It has a global view of flows in the network, clients connected

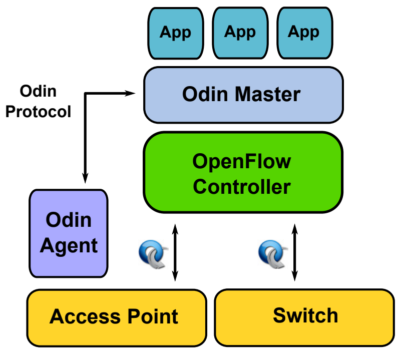

Figure 1: Architecture of Odin. The Odin Master (an OpenFlow application), speaks OpenFlow to the switches and the APs, and uses a custom protocol to talk to each Odin Agent running on APs. Odin applications use Odin's primitives to implement enterprise specific services.

to the network, and the infrastructure that comprises the network. Odin agents run on the APs. Together, the master and agents implement a Wi-Fi split-MAC (see CAPWAP [13]). A TCP connection is used between the agent and the master (established at boot time). It serves as Odin's control channel, and used by the master to invoke commands on the agents and collect statistics from them. Applications running on the Odin framework implement network services using interfaces exposed by the master.

2.2 Light Virtual AP (LVAP)

LVAPs are a central primitive within the Odin framework. When Wi-Fi clients in managed mode scan for APs, probe request messages are generated. APs responding with probe response messages become potential association candidates. A client then proceeds to perform a connection handshake with a locally selected AP. The LVAP abstraction enables to take control of association decisions from the client and leads to a logical isolation of clients with respect to the 802.11 MAC. With LVAPs, every client receives a unique BSSID to connect to, i.e, every client is given the illusion of owning its own AP. In Odin, the LVAP is the client specific AP. Each *physical* AP hosts an LVAP for each connected client. Removing a client LVAP from a physical AP and spawning it on another achieves the effect of handing off a client without the client performing a re-association, generating additional layer 2 or 3 messages, and most importantly, without requiring any special software or hardware at the client. Thus, Odin always provides clients a consistent link to the network, and the programmer of an Odin application needs not be concerned with *how* the client's link to the network changes. The end-point of a link always corresponds to the client IP and MAC addresses and the unique BSSID assigned by Odin. The implementation of the LVAPs, its resource utilization and beaconing issues are explained in Section 4.3. Note that LVAPs are considerably different from using virtual interfaces on a physical wireless interface (as is leveraged by OpenRoads [9]). With regular virtual interfaces, there is a state machine maintained for each virtual interface which the infrastructure has no control of.

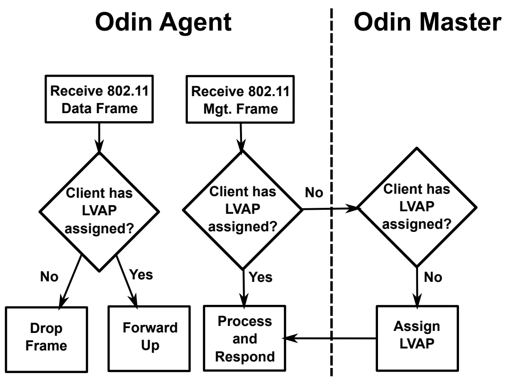
Odin Agent
Odin Master
Receive 802.11 Data Frame
Receive 802.11 Mgt. Frame
Client has LVAP assigned?
Client has LVAP assigned?
Client has LVAP assigned?
No
Yes
No
Yes
No
Drop Frame
Forward Up
Process and Respond
Assign LVAP

Figure 2: Processing path for 802.11 frames. When an 802.11 management frame is received by an agent, it checks whether an LVAP is hosted for the client that generated the frame. If yes, it processes and responds to the frame as per the 802.11 protocol. If there is no LVAP for the client, the agent informs the controller which assigns the client an LVAP. The agent then attempts to respond to the client accordingly.

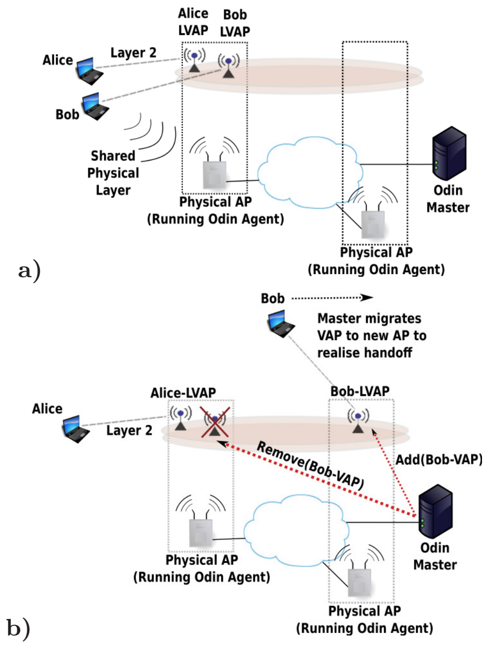
Alice LVAP
Bob LVAP
Alice
Layer 2
Bob
Shared Physical Layer
Physical AP (Running Odin Agent)
Odin Master
Physical AP (Running Odin Agent)
a)
Bob
Master migrates VAP to new AP to realise handoff
Alice-LVAP
Bob-LVAP
Alice
Layer 2
Remove(Bob-VAP)
Add(Bob-VAP)
Odin Master
Physical AP (Running Odin Agent)
Physical AP (Running Odin Agent)
b)

Figure 3: LVAP Abstraction: a) Clients connect to LVAPs, with one or more LVAPs being hosted on the same physical AP. LVAPs on top of a physical AP share the same wireless channel. b) The Odin master detects a client movement and performs an LVAP migration to realize a smooth handoff.

3. APPLICATIONS

In this section, we illustrate applications that can be built on top of Odin.

3.1 Seamless Mobility

With Odin's LVAPs, a handoff does not involve any additional message exchanges or state machine changes at the client. This can be leveraged by an Odin based application to implement mobility (Figure 3). The application can query various statistics provided by the Odin framework, and invoke handoffs depending on an application specific mobility metric. Our rudimentary mobility manager uses the receiver signal strength of the client at all agents that can hear it. The application ensures the client is handed off to the agent where this metric is the highest (Figure 4).

3.2 Load Balancing

For scalability reasons, the load at multiple APs can be balanced by dynamic re-assignment of clients to different APs (according to different metrics). Most existing work employs specialized software on the client, which the infrastructure uses to control client associations (such as [14]). The 802.11k amendment [15] also attempts to address this issue, but requires modifications to the client (or supplicant).

Since handoffs with LVAPs are inexpensive and fast, re-association based load balancing can be easily implemented as an Odin application. As will be explained in Section 5, even executing these handoffs every 100 ms does not result in any TCP perceived throughput degradation at the client.

3.3 Hidden Terminal Mitigation

Mitigating hidden terminals is a well known and classical problem in enterprise WLANs [2]. Several approaches exist to measure and mitigate the impact of hidden terminals in enterprise WLAN environments. They span from adaptive RTS/CTS to tight scheduling. However, most approaches require client modifications. An application mitigating hidden terminal issues perfectly suits Odin because of the centralized view of the network and the control over the association state of a client. This application is considered to provide per client measurements of link impairments (see [16]) such as hidden and exposed terminals and collisions.

We are envisioning further applications and extensions to Odin and LVAPs. In particular, full virtualization and slicing of the MAC and more measurement support with conflict graph construction.

4. ODIN FRAMEWORK IMPLEMENTATION

4.1 Odin Master

The Odin Master is implemented as an application on top of the Floodlight OpenFlow controller [17]. It uses an in-band control channel to invoke commands on the agents. In its current form, Odin commands can add and remove LVAPs and query for statistics. The master, through Floodlight, uses the OpenFlow protocol to update forwarding tables on the AP and switches. Odin applications execute as a thread on top of the Odin Master. Applications can view the statistics exposed by the master in a key-value format. As shown in Figure 4, applications can make decisions based on these statistics and use Odin's primitives to implement different services.

4.2 Odin Agent

Odin agents run on physical APs, and are implemented with Click elements [18]. Agents contain the logic for the Wi-Fi split-MAC and LVAP handling. Agents also record information about clients using radiotap headers, and commu-

```
while (1) {
  sleep (probe_interval);

  for client in odinMaster.getClients() {
    int max = 0;
    OdinAgent agent_to_handoff_to = null;

    for agent in odinMaster.getAgents() {
      // query agent for statistics
      Map<String, String> clientStats
        = agent.getStats().get(client.hwAddress);

      if (clientStats != null) {
        int signal = clientStats.get("signal");
        if (signal > max) {
          agent_to_handoff_to = agent;
          max = signal;
        }
      }
    }
  }
 // Perform LVAP handoff
 odinMaster.handoffClientToAp(client.hwAddress,
                              agent_to_handoff_to);
}
```

Figure 4: Pseudo-code of simple proactive mobility manager. The application LVAP-handoffs the client to the AP where the client has the best received signal strength.

nicate with the Odin Master over the Odin control channel. The physical AP hosting the agent also requires a slightly modified Wi-Fi device driver to generate ACK frames for every LVAP that hosted on the AP. For supporting authentication, the driver would also need a modification to inject per client encryption keys into the key table.

4.3 Realizing LVAPs

The first step to assign per-client LVAPs is to have Odin agents track probe request frames generated by clients. When an agent receives a probe request from a client for which it is not already hosting an LVAP, the agent informs the master. If the client is not already associated with the network, the Odin Master spawns an LVAP on the Odin agent which received the probe request (or one of the many agents which received the request). The agent then responds to the client's probe request with a unique BSSID provided by the master. Per-user network names (SSIDs) can be statically assigned, or a common SSID can be provided to all clients that attempt to connect to the network. To ensure logical separation between clients and their respective LVAPs, beacons (and probe responses) are unicast to the clients. This causes clients to drop all beacons that are not destined to it. Thus, each client can be presented with a unique BSSID to connect to. From this point onwards, the client connects to the newly spawned LVAP as in standard 802.11. An important implementation detail is to ensure that a client is consistently provided the same BSSID. In its current form, we statically assign BSSIDs to clients, but this is easily remedied by generating a BSSID deterministically for the client on the basis of its MAC address.

In its current form, an LVAP is represented by the following four-tuple:
$\{\texttt{mac\_address}_{client}, \texttt{ip\_v4\_address}_{client}, \texttt{lvap\_bssid}_{client}, \texttt{lvap\_ssid}_{client}\}$

This takes up a fixed size of 16 bytes, along with the length of the SSID. It is stored on the agent within a hash map keyed by the client MAC address. With a worst case length of 32 characters for the SSID, the storage involved on the AP with each addition of an LVAP is only 48 bytes. For this reason, hosting an LVAP does not incur a significant memory load on the AP. For each frame that the AP generates to a client, a lookup on this hash map is performed in order to obtain the right BSSID value to be used as the source MAC address in the 802.11 frame. Since this lookup is constant time and is not a function of the number of LVAPs, the processing load on the AP will remain a function of the number of packets the agent has to process (as is the case with a regular AP), and not a function of the number of LVAPs. Note that in order to support authentication, encryption keys need to be incorporated into the LVAP. This will further increase the memory used per LVAP.

However, since the agent generates beacons corresponding to each LVAP, having too many LVAPs can lead to more beacon generation, which in turn leads to more contention on the channel. Although we're yet to measure the impact of this on a client throughput, this can be easily remedied by using longer beacon intervals.

4.4 Network Authentication

The architecture of Odin is compatible with the two most deployed approaches to authentication.

WPA2 Enterprise: A client is authenticated against the system after it is assigned an LVAP. In WPA2 Enterprise, a client performs a series of handshakes with the AP and an authentication server in order to negotiate a session key. This session key is then used by both the client and the AP to encrypt the connection. With Odin, this session key can be accounted for in the LVAP state. Whenever any subsequent AP is to host an LVAP, it installs the corresponding session key into the wireless device key table. Thus, the LVAP approach is compatible with existing enterprise authentication schemes. We are presently working on implementing this.

Guest Wi-Fi: A client is assigned its own LVAP only after it has authenticated against the system. This fits with the common mode of authentication in guest Wi-Fi systems, wherein a user is given an unsecured Wi-Fi connection, and is redirected to a login page through the web browser [19]. The authentication mechanism can then return a token to the Odin Master for the client after successful authentication by the latter. The client is then assigned an LVAP.

5. EVALUATION

Since LVAPs are a central primitive of Odin, we perform experiments to gauge their effectiveness. The goal is understand what performance related assumptions Odin applications can make when using LVAPs. The testbed for our evaluation comprises a single client, two APs on the same subnet, and servers to run the OpenFlow controller and serve as a traffic end-point. All APs are based on a x86 architecture with Atheros AR9280 802.11abgn cards running Open vSwitch with OpenFlow 1.0 support. We are currently deploying Odin on a larger testbed.

In all experiments, the client is provided a static IP in order to exclude DHCP related delays. Authentication is disabled, and the standard IEEE 802.11 four-way handshake is performed when using Odin or legacy Wi-Fi. With Odin,

Table 1: Latencies involved in a layer 2 handoff using open authentication.

Scenario	Min (s)	Max (s)	Avg (s)	Std. Dev.
Same channel handoff	0.1737	0.3793	0.1897	0.0320
Different channel handoff	0.1767	18.3059	0.3632	1.0260

DHCP and authentication related delays only appear in the very first connection to the network, and not with each handoff. We run three experiments: In experiment E1, we measure the handoff delay involved in layer 2 in the 2.4 Ghz band over channels 1 to 11. This indicates of the delay incurred by the regular four-way handshake in a basic handover scenario. Note that Odin LVAP based handoff is not susceptible to this delay. Channel 48 in the 5 Ghz band is used for experiments E2 and E3 where the effect of LVAP handoffs on a TCP session are observed. In these two experiments, only the participating stations are transmitting on the channel. This ensures measurements under stable TCP conditions. All measurements are averaged over ten runs.

5.1 E1: Layer 2 Delay in Re-association

In this experiment, we measure handoff delays within a noisy wireless setting. All measurements are taken in a typical office environment during normal working hours. The client is made to associate with one AP, and then handed off to another. Using the `nmcli` utility, we observe the time delay involved for the client to disconnect from the first AP and complete its association with respect to the second one. The results are shown in Table 1. Handoffs are tried between all combinations of channels. The average handoff delay of 350 ms that we observe is in line with previous literature. The average delay for a handoff within the same channel is around 190 ms. As mentioned earlier, the delay for handing off within the same channel does not apply to Odin LVAP based handoffs. In Odin, a successful handoff occurs when a `REMOVE_LVAP()` and `ADD_LVAP()` messages are delivered at the old and new APs respectively. Since the master already has a TCP socket connection setup with each agent when the latter registers with the master, the time involved here is at most one RTT (assuming TCP is not forced to retransmit), along with the negligible amount of computation delay involved at both APs. In our testbed, this is less than 1 ms from the invocation of the LVAP-handoff call at the master (but can potentially increase if the network between the master and the agents is heavily loaded).

5.2 E2: Single Handoff Impact on HTTP Download

In this experiment, the client associates to one of the APs and begins a `wget` download of a large file. After 13 seconds, the client is made to handoff to the other AP. When using Odin, the handoff corresponds to an LVAP migration. In regular 802.11 mode, the client is explicitly made to handoff using the `iw` command. As explained above, the client uses a static IP and open authentication, which gives us a lower bound on the handover delays involved.

Figure 5 shows the throughput over time when using a regular 802.11 setup and having a handoff performed during a download session. The throughput drops to zero over

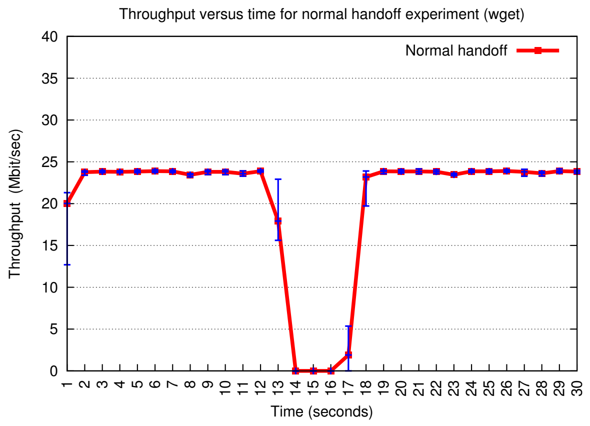

Figure 5: Effect of a regular 802.11 handoff without authentication or DHCP on the throughput of a file download over HTTP. There is a period of disconnectivity spanning several seconds when such a handoff is made.

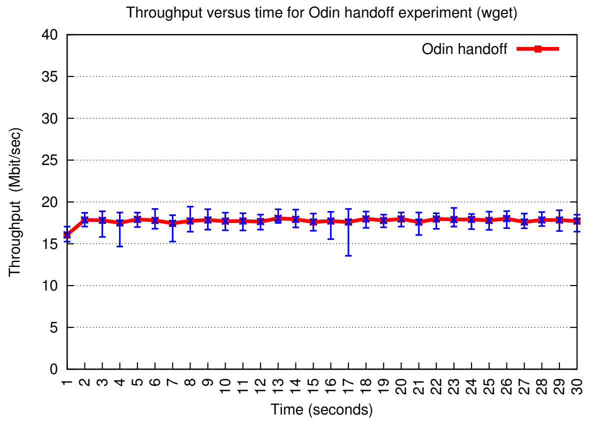

Figure 6: Effect of an Odin LVAP-handoff on the throughput of a file download over HTTP. There is no throughput degradation.

several seconds before recovering. In the presence of DHCP, and 802.1X authentication, this time window would be much larger. Figure 6 describes the same experiment performed with Odin and an LVAP-handoff.the throughput curve is unaffected in spite of the LVAP-handoff. However, Figure 6 indicates an overall reduction of throughput (close to 5 Mbit/sec) as opposed to the curve in Figure 5. Because we currently use userspace Click to run the Odin agents, slower and jittery forwarding performance on our AP hardware is obtained. Consequently, TCP is forced to throttle down. However, this is orthogonal to continuously maintaining layer 2 and 3 connectivity, which Odin achieves through LVAP migrations.

5.3 E3: LVAP-Handoff Benchmark

We want to understand how often an LVAP-handoff can be executed against a client without affecting its performance. The same setup as the previous experiment is used, but a single `iperf` based TCP session is executed with the client as the source. The duration of the measurement is 30 seconds. Between the 5th and 25th seconds of the measurement,

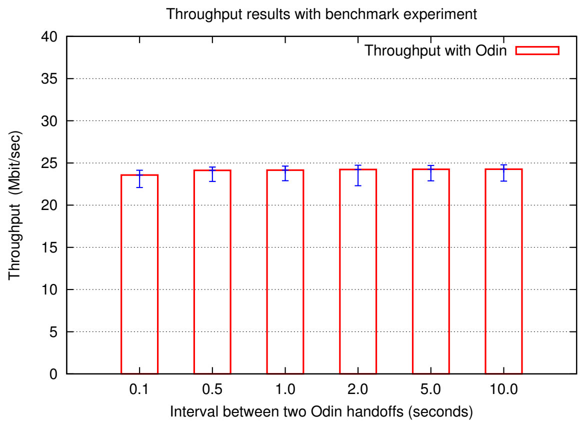

Figure 7: Average throughput achieved in the presence of continuous Odin LVAP-handoffs at fixed intervals. The throughput degradation negligible.

LVAP-handoffs are repeatedly made between the two APs at a fixed interval. We then observe the throughput degradation in this fixed interval. The results are shown in Figure 7. Despite repeatedly performing LVAP-handoffs every 100 ms, there is no significant throughput degradation. Although it is unrealistic for an Odin application to perform this many handoffs per second, it illustrates the inexpensive nature of this operation.

6. DISCUSSION & CONCLUSIONS

Odin shows the benefits of introducing programmability into the enterprise WLAN, by designing the right set of primitives. LVAPs are only one such primitive, which are lightweight and cheap and can be used to develop different kinds of network services efficiently. We are enhancing and deploying Odin on an indoor and outdoor testbed with a larger number of users [20]. We are exploring further abstractions in order to support the needs of a more diverse set of network applications. We also plan to explore fault-tolerance and fail-over capabilities and policy management for Odin.

7. REFERENCES

[1] *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999)*, 2007.

[2] 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. In *Proceedings of the 15th annual international conference on Mobile computing and networking*, 2009.

[3] R. Murty, J. Padhye, A. Wolman, and M. Welsh. Dyson: an architecture for extensible wireless LANs. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, 2010.

[4] R. Murty, J. Padhye, R. Chandra, A. Wolman, and B. Zill. Designing high performance enterprise Wi-Fi networks. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, 2008.

[5] P. Zerfos, G. Zhong, J. Cheng, H. Luo, S. Lu, and J. J.-R. Li. DIRAC: a software-based wireless router system. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, 2003.

[6] E. Rozner, Y. Mehta, A. Akella, and L. Qiu. Traffic-Aware channel assignment in enterprise wireless LANs. In *IEEE ICNP 2007*, 2007.

[7] P. Bahl, R. Chandra, J. Padhye, L. Ravindranath, M. Singh, A. Wolman, and B. Zill. Enhancing the security of corporate Wi-Fi networks using DAIR. In *Proceedings of the 4th international conference on Mobile systems, applications and services*, 2006.

[8] 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.*, 38:69–74, March 2008.

[9] K.-K. Yap, M. Kobayashi, D. Underhill, S. Seetharaman, P. Kazemian, and N. McKeown. The Stanford OpenRoads deployment. In *ACM WiNTECH 09*, 2009.

[10] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: a network programming language. In *Proceedings of the 16th ACM SIGPLAN international conference on Functional programming*, 2011.

[11] C. Monsanto, N. Foster, R. Harrison, and D. Walker. A compiler and run-time system for network programming languages. In *ACM SIGPLAN-SIGACT POPL 12*, 2012.

[12] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: A distributed control platform for large-scale production networks. In *OSDI 10*, 2010.

[13] CAPWAP protocol binding for IEEE 802.11. http://www.ietf.org/rfc/rfc5416.txt, March 2012.

[14] I. Papanikos and M. Logothetis. A study on dynamic load balance for IEEE 802.11b wireless LAN. In *COMCON*, 2001.

[15] *IEEE 802.11k-2008 (Amendment 1: Radio Resource Measurement of Wireless LANs)*, 2012.

[16] D. Giustiniano, D. Malone, D.J. Leith, and K. Papagiannaki. Measuring transmission opportunities in 802.11 links. *Networking, IEEE/ACM Transactions on*, 18(5):1516 –1529, oct. 2010.

[17] Floodlight. http://floodlight.openflowhub.org/, March 2012.

[18] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263297, August 2000.

[19] K.-K. Yap, Y. Yiakoumis, M. Kobayashi, S. Katti, G. Parulkar, and N. McKeown. Separating authentication, access and accounting: A case study with OpenWiFi. Technical report, OpenFlow 2011-1.

[20] T. Fischer, T. Hühn, R. Kuck, R. Merz, J. Schulz-Zander, and C. Sengul. Experiences with bowl: Managing an outdoor wifi network (or how to keep both internet users and researchers happy?). In *Proceedings of the 25th Large Installation System Administration Conference (LISA '11)*. Usenix.