VirtueMAN: A Software-Defined Network Architecture for WiFi-based Metropolitan Applications

Dimitris Syrivelis
Inst. of Telematics and Informatics
CERTH, Greece
E-mail: jsyr@iti.gr

Georgios S. Paschos
Lab. for Information and Decision Systems
MIT, USA
E-mail: gpasxos@mit.edu

Leandros Tassiulas
Inst. of Telematics and Informatics
CERTH, Greece
E-mail: jsyr@iti.gr

***Abstract*—Metropolitan scale WiFi deployments face several challenges including controllability and management, which prohibit the provision of Seamless Access, Quality of Service (QoS) and Security to mobile users. Thus, they remain largely an untapped networking resource. In this work, a SDN-based network architecture is proposed; it is comprised of a distributed network-wide controller and a novel datapath for wireless access points. Virtualization of network functions is employed for configurable user access control as well as for supporting an IP-independent forwarding scheme. The proposed architecture is a flat network across the deployment area, providing seamless connectivity and reachability without the need of intermediary servers over the Internet, enabling thus a wide variety of localized applications, like for instance video surveillance. Also, the provided interface allows for transparent implementation of intra-network distributed cross-layer traffic control protocols that can optimize the multihop performance of the wireless network.**

I. Introduction

WiFi is a pervasive technology with millions of networks deployed worldwide, nevertheless, used today mainly for providing 1-hop access of opportunistic nature. The reasons why the potential of the installed WiFi technology is largely unexploited are manifold: (i) operating in the free frequency band, WiFi technology is prone to interference and congestion, (ii) the management support of such networks is limited and in comparison to the cellular deployments they appear rather unplanned, (iii) deploying a multi-hop WiFi network is cumbersome and technically challenging if aiming to provide any Quality of Service guarantees, (iv) security is a nightmare.

A recent report from Cisco [1] reveals, that the growth in demand for mobile data, forces telecommunications providers to rethink the value of WiFi as an offloading option. Also, there are populous metropolitan areas like London or Seoul that are funding large projects for deploying WiFi-based free Internet access. Apart from installing new devices though, a huge benefit can be realised by enabling multi-hop communications over the existing deployments. In addition several studies [3],[4],[5] that analyze the performance and the user profile of urban WiFi deployments, identify that: (i) users tend to prefer these networks if they operate appropriately and (ii) many workload cases cannot be addressed efficiently by an unplanned and uncontrolled network deployment regardless of the available bandwidth. Employing a central controller to dynamically manage public WiFi deployments is a promising approach and there are successful products [6] which dynamically configure basic network-level operations at each access point, continuously monitor the deployed devices to alert the administrator and provide relaxed mobility support that relies on existing network attachment protocols.

In this work, we build on the concept of Software Defined Networking and propose a new architecture for deploying WiFi multi-hop networks. Our architecture has the potential to overcome the above challenges and enable a wealth of metro-scale modern applications that are based on mobile communications and social networking. The proposed architecture aims to provide (i) seamless mobility support, (ii) traffic filtering (iii) support for advanced Quality of Service schemes (iv) and facilitate the definition of private networks within the deployment without using the VPN overlay star topology. Our proof-of-concept implementation shows that pushing more network functionality at the access point devices and dedicating a few backhaul resources to centrally control it, enables a more efficient, scalable WiFi metropolitan network that can support many services beyond internet access.

The presented system is comprised of a novel *access point datapath* and a *central controller logic*. We propose a focussed, deployment-specific **datapath design** that features a series of lookup tables for access control, forwarding and QoS. The **controller** is used to manage and virtualize the network resources. It can be realized distributedly and configure the datapath on each access point via custom messages.

Our proposal attempts to address metropolitan WiFi challenges with the following architectural points: **Seamless mobility is facilitated for a large scale deployment.** There is no dependence on subneting and IP routing which complicates mobile operation.

End-to-end connectivity is maintained. For the case of IP traffic, mechanisms that break end-to-end connectivity, like for example NAT, are not needed.

Network access functionality is moved to the network edges. The Network Access Server functionality for the local clients is implemented at each access point rather than at the Internet gateway as it currently happens. This enables new network-level access and security schemes, that can be provided by the infrastructure network, for the direct mobile terminal interconnections.

Advanced resource allocation is enabled. The central control logic features extensions that enable end-to-end switching capacity reservation.

II. Architectural Context

In this section we give a brief overview of the architectural context via the description of the provided network-level services.

Forwarding Service. Intra-network forwarding is performed by entities called *edges*. The edge is often a wireless access point but it can also be an Internet gateway, a computer or any other network device that can be equipped with our protocol and perform routing. The clients of the network are assumed to be end-points which are attached to edges.

The forwarding tables are calculated by the controller in an offline fashion assuming that the edges are roughly immobile devices and the corresponding links are static. Then the forwarding tables are transferred/updated from the controller to the edges using a signalling protocol. Each edge is assigned a network-wide unique 2-byte identifier and features a local lookup table which associates all the deployed edge devices in the network with a forwarding record. Once the routing tables are deployed, a fast data path is formed between any two edges in the network. Using the lookup table, an edge can route data to any another edge in the network. The max edge identifier (0xFFFF) is reserved to implement network broadcast and the administrator may also define a range of identifiers that can be used for multicast.

The next step is to cater for client routing. Each client is attached to exactly one edge and has a client-specific identifier. The association of the client identifier to the edge identifier is is of fundamental importance to routing and it is performed by the controller. Then the latter distributes this information to several edges of interest; note that in case of virtual network, a client may be reachable only by a subset of edges that actually participate in the virtual network. A network function is developed to read the destination client identifier, lookup a local table to determine the associated destination edge identifier, assign a QoS value and encapsulate the packet for forwarding. Although the edges are assumed to be static, clients attach and detach to/from edges continuously and the corresponding client routing information must be constantly updated. An optimised signalling protocol is used for this task, to distribute association information in the form of configuration updates. The described operations are supported by a simple 2.5-layer protocol is used.

Network-level Authentication Service. The authentication logic resides in the controller and the authentication actions are performed by each edge. When a client joins the network for the first time it is assigned by the edge a *temporary* network configuration. The only available service with the temporary configuration is the network authentication server. After the client provides the necessary credentials, the controller confirms a permanent assignment of a network configuration and updates the necessary routing entry for this client so it becomes reachable. The edge performs a packet-by-packet authentication.

Mobility Support. First note that the commonly used WiFi access point 802.11 association algorithm is a first-layer approach to mobility already supported by current systems if the same SSID is used by all the access points in the network. In this section we describe how our proposed architecture deals with the challenge to keep the network aware of the whereabouts of the mobile clients so that they remain reachable by any other client in the network.

Reactive approach: The basic supported functionality is of reactive flavor. Following a coverage area transition, the authenticated client gets associated to a new edge and the first packet that it sends to the network triggers a mobility event that is immediately delivered to the controller. Then custom messages are used to move the client-specific information (record entries) to the new edge and also notify the edges that serve terminals which were connected to the moved client. This is a rather slow solution, which, however, is immediately supportable by the current architecture and guarantees the mobility of clients.

Proactive approach: In order to boost mobility performance and efficiency, a proactive scheme is also employed by the network. In this case, client-specific records are maintained at an administrator-defined number of geographically neighbouring edges. When a client moves to a neighbour edge the destinations/QoS lookup table is already in place and the controller proactively sends a copy to new neighbours. Taking advantage of the source identifier field of the forwarding packet each involved edge can infer locally the mobility event for any client with active outgoing connections to the moved target, thus improving the response performance of the network.

Network-wide Quality of Service Support. The quality of service network function is based on a queue-based scheduler. A QoS 2-byte value record is added in the packet header and used to differentiate service among flows. Each client is assigned QoS values for each flow and this assignment is stored in the host access point in the form of client-specific destinations/QoS lookup table. QoS algorithms that are currently used in the proposed network are beyond the scope of this work.

III. Architectural Blueprint

In this section a blueprint of the proposed network architecture is presented.

Programmable Datapath. To provide the network services described above, every edge device features a programmable datapath, which is depicted in Figure 1 in the form of boxes representing packet processing stages along with the directed interconnections that imply the execution sequence. Following the SDN paradigm, the datapath features a series of lookup tables that configure all packet processing and forwarding operations primarily in a proactive manner, but also reactively to network events when needed. Below, we explain the different interconnected operations given in the Figure 1.

The two main I/O points are the *backhaul driver* module and the *Client WiFi Bridge*. The backhaul driver receives packets over the infrastructure network while the bridge connects the edge with different clients. When a packet arrives from the

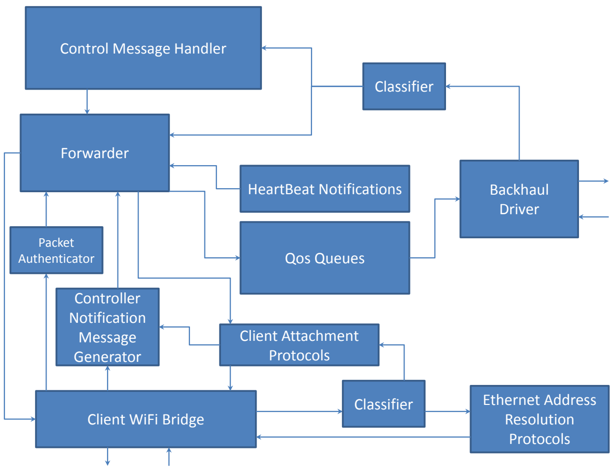

Fig. 1. Edge Programmable Datapath Design: All packet processing stages of the datapath are depicted. The directed interconnections represent packet transfers between the stages and imply the processing execution flow. All stages access lookup tables to perform operations, that get configured in the Control Message Handler stage

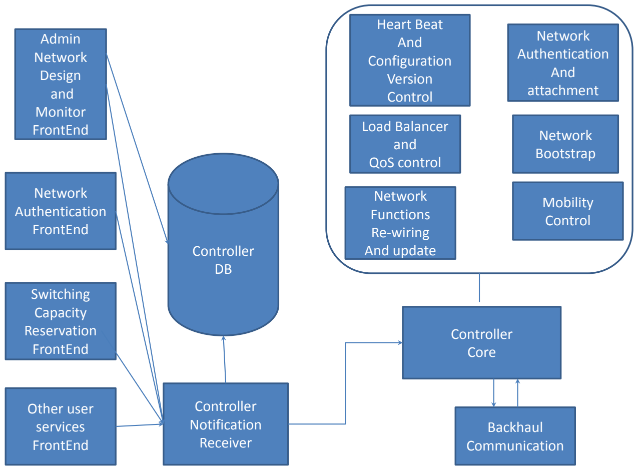
Admin Network Design and Monitor FrontEnd
Network Authentication FrontEnd
Switching Capacity Reservation FrontEnd
Other user services FrontEnd
Controller DB
Controller Notification Receiver
Heart Beat And Configuration Version Control
Network Authentication And attachment
Load Balancer and QoS control
Network Bootstrap
Network Functions Re-wiring And update
Mobility Control
Controller Core
Backhaul Communication

Fig. 2. Controller Subsystems Overview

backhaul driver, it is first classified as control or data packet by inspecting the forwarding header type field. The control messages are processed by the *Control Message Handler* function - all functionality related to receiving control messages in an edge is implemented there. The *Forwarder* interconnects all the datapath operations because it supports forwarding - a core network function. The forwarder is comprised of a collection of networking functions that perform different operations per input. More specifically, the forwarder receives: (i) client data packets from the local WiFi bridge, (ii) controller notification messages that support mobility, (iii) new client attachment notifications or (iv) Heartbeat messages that notify the controller of the current node liveness, usage statistics and installed configuration version.

The *Packet Authenticator* features one network function per input to support control for different client network layer protocols.

The *Ethernet Address Resolution Protocol* features functions to return a globally agreed (for mobility), fake ethernet address to support typical link-layer attachment. The *Network Attachment Protocol* functions bridge the client-specific attachment with a unified attachment protocol that is supported by the controller. Finally, the localhost netlink socket is used to support datapath software updates.

Controller Design and Operations.The proposed network controller is comprised of several subsystem types which are depicted in Figure 2. Administrators and users interact with a variety of *front-end* interfaces that export all available functionality. The administrator control front-end is tightly integrated with the *controller database*. All the front ends communicate with the *controller notification server*.

Network deployment. A network design web based tool has been developed that facilitates the deployment phase of the network. Initially, the administrator must register the edge devices that comprise the network. The tool is used to register the backhaul interface addresses, to assign the edge identifier, to add the GPS location and define all the one hop neighbours. In the sequel it produces a configuration file for each device that needs to be stored either on device flash memory or external usb. This configuration information is only required during low-level device bootstrap which enables it to receive and send controller messages over the backhaul. The tool stores all this information in the controller database and computes offline the network graph. On the produced graph, the tool computes the destination shortest path tree for each node in the network and subsequently builds the backhaul forwarding tables for each edge device. Moreover, in order to efficiently support broadcast, the minimum spanning tree of the network graph is also calculated and appropriate forwarding entries for broadcast address are added in each device forwarding table. Having all this information available the network bootstrap process can be initiated. The controller floods the configuration in the network, i.e. firstly sends it to the one-hop neighbours, after those are configured, they can be used to forward to the 2-hop neighbours that remain to be configured and so on until all the network is configured.

Authentication. The controller supports the authentication service by exporting a specific user front-end that uses some authentication gateway e.g. kerberos, LDAP etc. Following an authentication request, the controller responds with a message that updates the lookup table of the corresponding edge to which the client is attached.

QoS. Resource allocation mechanisms are used both for end-to-end bandwidth reservation as well as for the definition of private networks within the deployment. The resource allocation mechanism is built in two steps: (i) in the controller, a mapping of clients to QoS classes is designed and (ii) in the data path the classes are mapped to specific scheduling actions. The administrator may alter the allocation policy; defining a new QoS configuration, the controller will configure the access points where the source users are attached, by adding the QoS value to each terminal-specific destinations/QoS table.

From the client point of view, the QoS mechanism works as follows. When a new client is attached and authenticated to the network, only default destinations with default QoS values are available. For the rest of the destinations, information is not available and forwarding is denied. The user will have to access the controller front-end that provides information on who is connected and what services are available.

Moreover, the controller can implement dynamic load balancing schemes for popular destinations, like itself and Internet gateways. This support is implemented efficiently due to the detailed control of the forwarding. We give an example: consider a network with many IP Internet gateways attached to different edges in the network, all having the same IP. After observing the current load, the controller can dynamically configure forwarding on selected access points to use different destination edges to reach the gateways, so the traffic can be redirected transparently and balance the load across different backhaul links. What is more important, users may exploit this support to provide services to the network like sharing their Internet connection using a time-based scheduling policy of their choice.

Configuration versioning. The Heartbeat protocol and the configuration versioning are very important controller features that support the integrity of the in-network state. Configuration versioning is realised separately for each network edge. The datapath on each device sends out periodically a Heartbeat packet, that contains the current configuration version identifier, the local QoS queueing system weights and usage statistics for each backhaul interface. Upon reception, the controller logs the information and performs any actions if needed. If the heartbeat configuration version does not match the latest version that the controller has logged for that edge, then the timestamp delta between sending the last configuration message and the heartbeat arrival is compared against an estimate of the backhaul latency for that destination. If the latency estimate is exceeded, the configuration packets are resent to that access point to guarantee delivery.

Power failure and versioning. The network edges store the main forwarding table in non-volatile memory so in case of a power failure the communication with the controller is possible after the restart is completed without the need of a new initialization process. In the event of such a failure, the next heartbeat message carries a special notification to the controller, that of a system restart event. Subsequently the controller starts a new configuration version sequence, determines the current active users and resends configuration packets. If an administrator-defined number of heartbeat messages do not arrive as expected, the device is considered dead by the controller and an alert is issued.

Communication Channel and Message Types.The backhaul liks are used by the controller for dissemination of control messages. Using the QoS mechanism, control messages are assigned the maximum priority so that it is highly unlikely that a configuration packet is dropped or waits too long in a queue in the network due to congestion. Nevertheless, we require 100% integrity of network state. Since configuration packets are sent frequently, we do not use a dedicated acknowledge scheme with reverse messages; the acknowledgment is implicitly conveyed by the Heartbeat mechanism. The communication channel does not employ any encryption and relies on backhaul-level security, but it will be considered in future work. In the current design all controller messages should not exceed the packet MTU because fragmentation is not supported. On the other hand, the protocol supports message aggregation in a single packet for proactive configurations. Regarding the message format, the first 2 bytes of all control messages carry the message identifier, while the rest of the message is customised per operation. In the forwarding header the controller messages are designated by the payload type byte being equal to 1.

Firstly the messages classified as controller-to-datapath messages and datapath-to-controller messages. The messages initiated by the controller include (i) forwarding lookup table add/remove operations, (ii) add/remove table entry actions for authentication, (iii) generic network attachment response message, (iv) QoS table operation and (v) a tunnel message type for transferring new function configuration and/or updating datapath module binary. The datapath generates 3 different message types, (i) the new node attachment/authentication request, (ii) mobile node attachment requests and (iii) Heartbeat messages.

IV. Implementation

The programmable datapath has been implemented on the *click* modular router packet processing framework[7]. In *click* architecture, the developer builds packet processing functions which are integrated in a single binary. Execution sequence of functions is defined by a configuration file that is loaded during *click* bootstrap and, with certain limitations, a different configuration file can be loaded at runtime. *click* features have been exploited to define network function interfaces and support dynamic re-wiring of the execution sequence. The Network Functions Virtualization concept is thus enabled; a developer may use the *click* environment along with the message building library and the defined packet format interfaces of the programmable datapath, to add/replace remotely network functions and the administrator may enable different versions of network functions to be invoked for the same operation on demand. Note that target edges are resource constrained devices that do not support the required instruction set extensions for virtualization but this is likely to change in the near future. Therefore, we are considering to exploit the click-based approach described in[8] to better support Network function Virtualization which just requires a straight forward port of the current codebase.

For the egress path towards the backhaul network, 4 lookup tables need to be visited before a packet gets forwarded: (i) the authentication lookup table, (ii) the mobile terminal destination address/QoS lookup table, (iii) the destination client-edge association table and (iv) the edge identifier lookup table for the next hop towards the destination access point. The ingress path just needs one lookup to the authentication table to determine destination link layer address for the designated client. We have implemented all lookup tables and respective network function versions to support the IPV4 protocol for the network clients. Regarding the IPV4 network attachment we have implemented a custom, controller-assisted DHCP relay protocol.

The controller core has been prototyped in python and relies on a mysql database to support all operations. The controller can be deployed in a distributed manner by running copies of the core in the network. Taking advantage of the network forwarding control architecture, additional controllers may be deployed, even during network operation. Each controller copy works with a local database which is synced every time a new network event takes place e.g. new node attachment. In order to not sacrifice mobility response performance, all controller copies can send configuration changes to all edges instantly and sync databases later. Due to this operation a race condition is introduced to the heartbeat protocol which results in delay in case a packet is lost, a small price to pay compared to the response benefits. Regarding QoS we used a simple class-based scheduler and we intended to address more efficient approaches in the future.

A. Target Platforms and Experimental Deployment

The programmable datapath is running as a *click* kernel module in linux kernel version 2.6.38. The datapath has been installed on 2 different platforms: (i)the inexpensive ALIX 2D2 board that features a single 500Mhz AMD Geode CPU and 2 wireless interfaces over mini-pci with Atheros 5213 chipset that support 802.11g and (ii) the a high performance Intel atom N2800 cedarview which is dual core and runs at 1.86Ghz.

We have deployed the described network in the University building with 5 Intel atom access points and we have conducted some early experiments. For the backhaul links we used one dedicated wireless interface per access point and all devices were connected in a row, serving five different floors of the building.

B. Experimental results and scalability issues

The datapath performance was measured on both devices in terms of throughput and CPU usage. This is an important measurement that shows that our architecture can be installed in existing low-end targets. To push the system to extreme operation, we made throughput measurements where the access point forwarding table is populated with 65535 entries and the rest of the tables with 1000 entries each, thus reflecting the operation on a large metropolitan network with 1000 local clients connected. The throughput was measured over wired Gbit Ethernet to test the capabilities of the low-end device to push packets. Using two wired Ethernet interfaces, we connected a client laptop to the first and the second was used as the backhaul to a server. The source iperf instance was running on the laptop and the destination iperf on an icore7 desktop which was also running a local datapath to support the controller. The Geode platform achieved 83 Mbits/sec with the processor working at 100%. The intel atom platform achieved 536 Mbits/sec with just a single core fully loaded. In both cases the bandwidth is sufficient for supporting both wireless interfaces that operate simultaneously at full speed. In practice the Geode processor needs to operate at full speed to support the achievable wireless throughput, while the atom platform did not exceed the 70% of the single core capability.

The deployment was connected to the Internet via one gateway and credentials were provided to 87 graduate students and staff of the Department. The Heartbeat period was configured at 1 second. We have tested all described services and during peak hours the controller had to serve 35 messages per second at an average. The next step was to stress the controller by exploiting the heaviest operation which is to retrieve and send a client configuration to an access point. The database was loaded with 100.000 entries in that table and fake mobility notification messages were massively issued. The icore7 platform could serve 650 requests per second. For the mobility scenarios the proactive approach was completed in a few hundreds of msecs and active TCP sessions were not affected seriously during the transfer. The reactive approach never exceeded 2 seconds. These response times were measured concurrently with average traffic load generated by users in the offices. Taking into account that a controller can effectively serve 650 requests in a worst case scenario, a single controller instance may support 1615 users (taking into account that 87 users generated 35 requests/sec). Note that these are early results and the scalability of the approach is an issue that will be considered in future work.

V. Conclusions

In this paper we presented VirtueMan, a WiFi-based metropolitan network architecture that enables thorough management of the deployed network resources. This level of control guarantees the user experience, provides improved security and allows for the effective deployment of mission critical services like metro-wide video surveillance systems that can take over the network to support a crisis. In the future we plan to address scalability issues of the proposed approach.

Acknowledgment

This work has been supported by the European Commission through the FP7 “CODELANCE” program, under grant agreement No. 285969 (IAAP).

References

[1] S. Taylor, A. Young and A. Noronha, “What Do Consumers Want from Wi-Fi?”, Insights from Cisco Internet Business Solutions Group (IBSG) Consumer Research, May 2012.

[2] IETF RFC 6275 -Mobility support for IPv6.

[3] M. Afanasyev, T.Chen , G. Voelker and A. Snoeren, “Analysis of a mixed-use urban wifi network: when metropolitan becomes neapolitan”, Proceedings of the 8th ACM SIGCOMM conference on Internet measurement, 2008

[4] V. Brik, S. Rayanchu, S. Saha, S. Sen, V. Shrivastava, and S. Banerjee, “A measurement study of a commercial-grade urban WiFi mesh”, In Proceedings of ACM Internet Measurement Conference, Oct. 2008.

[5] D. Tang and M. Baker, “Analysis of a metropolitan-area wireless network. Wireless Networks”, 2002.

[6] Ubnt networks Inc, “Unifi Controller”, http://www.ubnt.com/unifi

[7] E. Kohler , R. Morris , B. Chen , J. Jannotti , M. Frans Kaashoek, “The click modular router”, ACM Transactions on Computer Systems (TOCS), Aug. 2000

[8] M . Ahmed, F. Huici, Felipe and A. Jahanpanah, “Enabling dynamic network processing with clickOS”, SIGCOMM Comput. Commun. Rev., Oct 2012