***Abstract*—Contemporary network schemes are built around the use of proprietary networking devices interacting with general autonomy in the area of sending and forwarding network data. This paradigm presents a handful of issues, including a lack of centralized intelligent network transparency in data logistics, and a lack of centralized, intelligent, network control. These devices are controlled by the closed-source firmware of their manufacturers so researchers and network administrators are unable to implement experimental routing rules. However, using software abstractions and management applications, the network can be controlled centrally, and treated as a whole. This is the concept of software defined networking. Wireless ad hoc networks present similar problems. An ad hoc network has no centralized hardware or control and may be made up of heterogeneous nodes. We have adapted ideas from software-defined networking and applied them to a wireless ad hoc network to create a modular ad hoc network management structure that can be easily modified and extended, including an implementation using Android smartphones. Interfaces allow components to communicate without knowing how other components work. Third-party applications can use the interfaces to access the ad hoc network, reducing development time and program complexity.**

***Index Terms*— Software-Defined Networking (SDN), Ad hoc, MANETs, Android.**

1. Introduction

Mobile ad hoc networks (MANETs) consist of mobile devices (nodes) that are capable communicating with each other to form a wireless network without any extra infrastructure such as routers or switches. MANETs have been widely studied for both commercial and military applications. Nodes in a MANET are able to move freely, so the topology of MANETs can change at any time. Additionally, MANETs may consist of heterogeneous node. Heterogeneous MANETs introduce additional challenges as different devices may require changes to the MANET software at both the communication and application layers. Many large infrastructure networks suffer from similar challenges because they may be made up of hardware components produced by different manufacturers, each with their own closed source firmware. To address this problem in infrastructure networks, the concept of software defined networking (SDN) has been proposed, which we apply to ad hoc networks.

In software-defined networking, device configuration, network state, and the network-level communication is abstracted into three distinct components: the hardware layer, network operating system (NOS), and application layer control programs [1]. The hardware layer consists of the physical network devices like routers and switches. The network operating system controls the network based on information about the current state of the network. The application layer control programs provide management and security controls by interacting with the NOS. These components interact through interfaces which allow different configurations of components, making it simpler to support different types of hardware.

By applying these concepts to ad hoc networks, similar benefits can be achieved, improving the utility of ad hoc networks and reducing development for both applications using the network and routing protocols. Our proposed structure is made up of the same three components that SDN is comprised of. We replace the hardware layer used in SDN with a more general networking layer that handles both network communication and routing, using the Ad hoc on demand distance vector (AODV) routing protocol [5]. Ad hoc routing protocols are not controlled by higher level components directly and always function exactly as designed to prevent unpredictable network behavior and potential routing problems. Currently the NOS layer provides a view of the network, handles sub-networks for each application using the system, and allows custom routing rules. The network map allows higher layer applications to know what neighbors are available without interacting with lower level routing algorithms. Control programs may use it to modify the way data packets are forwarded without changes to the routing protocol. The components interact through a simple interface, which allows them to be replaced or modified without changes to the components running on other layers. In addition, third party applications may use the interface to interact with the network via the NOS, significantly reducing the difficulty of developing applications that use the ad hoc network. Both SDN and its application to ad hoc networking is described in more detail in the following section.

Our system has been implemented using the Android operating system and tested with an ad hoc network of Nexus One smartphones. To enable ad hoc networking, the smartphones were rooted, but no other permanent modifications are required, so the system should be able to be adapted to most devices running versions of the Android 2.3 operating system. To enable easy modification, all routing is done in the application layer in Java at the cost of performance. Specifics of the implementation are discussed in section III.

1. Methodology

This section provides a summary of Software-defined networking and then explains how it was adapted to work in an ad hoc network.

* 1. *Software-Defined Networking*

Software-defined networking was proposed as a way to improve current networking infrastructure by making it more dynamic and easily configurable to meet the demands of current trends in computing such as virtual machines and cloud computing, which use existing hardware for adaptable software [1]. SDN approaches the problem by separating the data plane and control plane with a series of abstractions. First, the data plane is separated from the control plane by creating the first layer of a software-defined network: the infrastructure layer.

The infrastructure layer includes all the physical networking devices like routers and switches. By removing the control plane from the network devices, they can be simplified because they no longer need to deal directly with the protocols and can simply respond to the layers where the control plane is moved to. The control plane is further divided into layers.

The network operating system tracks the state of the network and its usage. Control of the network is handled by application layer control programs which utilize abstractions provided by the NOS. All of the layers communicate using interfaces to abstract the functionality of the other layers, allowing them to be easily modified to suit the needs of the users and network managers.

 **

Fig.1*.* Traditional software-defined network comprised of switches and servers, with end-user devices.

* 1. *Applying SDN to Ad hoc Networks*

In order to apply the traditional concept of software defined networking to our ad hoc network of smartphones, certain design modifications had to be made. This is due to the fact that, unlike the physical structure of a traditional network, an ad hoc network contains a constantly varying set of similar nodes -- dedicated servers are not a natural component of such a peer-to-peer network. Additionally, this means that several functions and purposes are attributed to the same device.

This necessitates the implementation of various communication protocols, to be used for communication between abstraction layers on the same device, and across devices. Inter-process communication needs to be implemented between the control application and the network operating system, as well as between the network operating system and the packet forwarding layer. Additionally, there needs to be an inter-process communication method between the network operating system and external applications on the local device. After that, networking protocols must be developed which will allow the network operating system instance on one device to communicate with instances on other devices. These communications include commands forwarded from the control application, various network operating system functions, and data transmissions initiated by external applications.

These inter-process protocols must be well designed and clearly defined to preserve the adaptability of the system. The network operating system should be able to pair with any packet-forwarding layer software which implements the specified protocol. The coupled system should also remain compatible with other nodes, regardless of the implementation of their packet forwarding layer. The packet forwarding layer protocol is similar to OpenFlow in its instruction set and adaptability [2]. Similarly, the protocol by which the network operating system communicates with the control application must be well defined, such that alternate control applications can be used over the network.

One of the prerequisites for meaningful, centralized, network control is having a global view of the network. This global network view will be used by the network operating system and control applications to make routing decisions and rules. Choosing an algorithm for network mapping is significant, as the ad hoc network is constantly restructuring. The mapping algorithm should be able to build an initial graph of all the network nodes, associating nodes which can communicate directly, without forwarding packets through a middleman. Map management software, running within the network operating system, should then occasionally poll adjacent nodes to ensure that they are still within range. Should a node leave range (that is, cease to be accessible via a single hop), a network update message should be broadcasted, notifying others of the change. A similar process should be executed when a node enters the range of another.

The nature of ad hoc networks also presents an obstacle to the design of the control application. In traditional software-defined networking, such as in an OpenFlow network, the control application runs on a server connected to the network [2]. It runs as a single instance, on a single computing device, and communicates to the packet forwarding layer via distributed instances of the network operating system. In an ad hoc network, however, it is unrealistic to believe that there will always be some dedicated server within range. It is a network comprised primarily, or entirely, of end users and their personal devices. So where does the control application run? One option is to allow the control software to run on any or all devices. This is the simplest solution, but leads to issues, such as when two separate control programs try to publish conflicting rules. Additionally, if the network operating system needs to question the control application, which instance of the control program should it consult? A better solution is to dynamically assign a network host device for the control application. This could be the node with the quickest average connection speed to its peers, or the node which adds or drops connections least frequently. Whatever the algorithm is for choosing a control server, it is important to frequently broadcast state information to a set of backup control servers, which can be reverted to if the connection to the initial server is lost.

Over the ad hoc network, the control program is responsible for defining the network’s routing habits by dictating specific rules to the network operating system. This includes the ability to add and remove flow table entries, and to set rules for specific flows. It should also be able to respond to some set of network operating system requests, such as where to send a packet which matches no known forwarding entries.

We also found it useful to support the definition of metadata for nodes. For example, in an ad hoc network of android smartphones, it is not very intuitive to expect end users to recognize each other’s devices using solely their IP addresses. Some other tag, such as a user name, associated with that IP address can give end users a better understanding of which other user is attributed to some network node. This construct can also hold additional user information, if desired.

Virtual network slices enable the virtualization of multiple data networks over the same physical ad hoc network. An implementation of network slicing markedly convenient for an ad hoc network of smartphones is one wherein each external application runs on its own network slice. This enables developers and users of the network to very easily find peers running similar software. Also, it provides a very convenient method of implementing application-specific routing rules, as the control application should be able to manage each of these slices individually.

It is the responsibility of the network operating system to maintain information about the various virtual network slices. Each slice should be given some unique identifier, and maintain a set of member nodes. Additional properties could include, for example, a speed-vs.-reliability index for connection preference. The network operating system could use this index to choose a suitable path for routing data on that network slice. The virtual network definition is very open and flexible.



Fig.2. In our software-defined ad hoc network, each device acts as a packet forwarding node, runs an instance of the network operating system, and serves as an end-user device running user software.

1. Implementation

This section describes the implementation details of our system. Each component is described, starting from the network layer and going up to the application layer, including descriptions of the interfaces between components. Our implementation is a simplistic demonstration and test of the design we have developed. The functionality of this implementation is merely a subset of the capabilities of the design, used to determine the feasibility of the design’s core structure.

* 1. *Network Layer*

While traditional software defined networking uses network devices, whose default routing protocols are hidden in firmware, via a manufacturer implemented standard (such as OpenFlow), our implementation relies on an ad hoc service written in the application layer [1]. The Wi-Fi ad hoc service is a modified version of the one created for the Nest (NEtworked Smartphones for Target Localization) system, which uses a process running C code configure the operating system’s Wi-Fi adapter [3].

The Ad hoc On-demand distance vector (AODV) routing protocol was selected due to its simplicity and well-studied performance [4]. Each node in AODV tracks its neighbors through the use of Hello packets, which are periodically broadcast. When a route is desired, a route request (RREQ) is broadcast throughout the network. When a route to the destination is found, a route reply (RREP) is sent back to the initiating node, updating routes to both source and destination for intermediate nodes. When a node loses a neighbor a route error (RERR) is sent to nodes that rely on the node’s connection with the lost neighbor for routes [5]. The default AODV routing protocol was left intact, and is used to generate the default ad hoc network structure. All of our software-defined rules would modify this default structure, leaving the actual route tables alone to prevent inconsistencies. The application was modified to run as a background service, which the network operating system interfaces.

* 1. *Network Layer – Network Ooperating Ssystem Interface*

For inter-process communication between the three management layers, we used the Android Interface Definition Language (AIDL). This protocol uses simple user-defined interfaces to generate code, which can be used to emulate calling methods directly on one application from another [6].

[Interface specification diagram]

* 1. *Network OS*

The network operating sends external application data through the sendGenericPacket(…) interface method, and maintains our implementations of both network mapping and network slicing with the logical network methods as described below.

 To prevent the removal of connections necessary for network maintenance, as well as for accurate network mapping, removed entries are actually added to a block list. This list is scanned, and entries within it are ignored, by default. Some packet types, however, are allowed to ignore blocked entries. This includes mapping packets, as well as configuration request packets.

* 1. *Network OS – Network Mapping*

Our map management software is, accordingly, rather simple. The algorithm used to generate a network map is simply to recursively scan the network every time map data needs to be refreshed. The node initiating the refresh sends a map request packet to all single-hop peers, including; packet data unit type, a randomly generated integer id, the node address of original requester, a blacklist of already visited nodes, and a global view table. The first node labels itself as the original requester, adds itself to the blacklist, and adds a series of pairs to the global view table – each pair containing the original node itself and one of its single-hop peers. It also records the id of the request locally, marking it as its own. It then sends copies of this packet to all of those single hop peers, which each receive it, add pairs representing their single-hop peers, blacklist themselves, and continue forwarding to peers which are not already blacklisted. These nodes also record the id of the packet, instead tagging it locally as completed. When a node receives the same map request id twice, it instantly sends the packet back to the node which initiated the map refresh. Likewise, when a node receives a map request which it cannot forward because all its single-hop peers are blacklisted, it sends the completed request to the original requester. That original node collects all of these responses, recognizing the request id as one that it originated, and uses the received data to build an adjacency list, representing the network.

This process is repeated every time map data needs to be updated. A more refined implementation would include persistent monitoring, and simple triggers to add and remove individual connections from the network map, without a full rebuild.

* 1. *Network OS – Logical Networks*

A logical network is the name we have given to our implementation of network slicing. Each logical network running over the physical ad hoc network is distinguished primarily by its logical network tag – some String naming the logical network. The idea is that each external application would run in its own logical network, making it easy for the end user to find peers running similar software, and also offering a convenient method of implementing application specific routing rules.

Our logical network implementation involves the use of a logical network manager in the network operating system, which maintains and simplifies access to a set of logical network objects. All external application data sent through the network must be done via a logical network object, which currently defines only a network tag, and set of integer node members.

* 1. *Application Layer – Control Program*

For our small scale demonstration implementation, we used a very simple control application structure. It resides on all network nodes, but with only one instance used during test trials. In the case of a conflict, the most recent request would be executed. The control application communicates with the network operating system using the INetworkOS AIDL interface.

The control application initializes the network if it is not already connected by calling startWifiService(…) on the network operating system. It passes information collected from the user to configure the ad hoc network it will join. The control program’s interface provides information about the current network state, such as whether or not the network operating system is connected to the Wi-Fi ad hoc layer; whether or not the Wi-Fi service is running; or if the ad hoc routing protocol is active or not. Once all of the system components are running, the user is able to enter network management.

The network management portion of the application allows the user to view the global network map and to update it by adding routing rules. The network map is displayed as an adjacency list. Packets can be sent to provide a route trace to other nodes in the network. The ad hoc service displays a user interface message each time a tracing packet is forwarded, enabling us to visually assert that any routing rules are being adhered to.

The control application can also display a contact’s forwarding table, where routing rules can be added or removed. Each forwarding entry shows the details of that entry. These details include precursors, validity, next hop address, destination address, hop count, and destination sequence number. Some of these details are specific to AODV [5]. A different routing protocol would need to provide different forwarding entry details.

Routing rule modifications are carried out by configuration request packets. Each request encapsulates a list of RouteTableEntries, a ConfigurationRequestType, the requesting node id, the target node id, and a logical network tag. RouteTableEntry is the abstract data type that we use to represent forwarding table entries. ConfigurationRequestType is an enumerated type that consists of the following:

Our control program implementation is designed primarily for concept testing and demonstration. A more advanced control program could configure the routing protocol used by the system or could manage security in the network.

* 1. *Application Layer – External Applications*

A significant challenge encountered was choosing how to interface external applications. We wanted a simple and quick interface, which developers could learn and use with little fuss. The ultimate objective was for the external applications to supply the network operating system with a destination node ID, and a byte array of data. It would be the responsibility of the external application on the foreign device to interpret the byte array data. The two methods considered were AIDL, and the Android Intent system [7].

Both methods were implemented and tested. The intent based system requires third-party developers to implement an Android BroadcastReceiver, which listens for intents broadcast by the Android operating system [7]. Data sent in an Android intent is provided as a key-value pair, so developers need to know the keys used for the various Intents used by the network operating system. A small library was developed to simplify the use of AIDL for third-party developers. The library sets up an AIDL connection to the network operating system which automatically connects to the network operating system. The library then provides a handle to a method in the application that will be called by the network operating system when data is received for the application. The library allows the application to view other nodes in the network running the same application, to send data to remote devices running the application, and to view the node’s own contact information. The library could be further modified to notify the application when nodes join or leave the network.

Two demonstration external applications were implemented: a messenger application and a file sharing application. Both applications work in essentially the same way. The applications broadcast their desire to join a logical network, including a tag representing the application, or create the logical network if none exists. The network operating system uses this broadcast to add the node to the logical network, or to create the logical network.

1. Results
2. Initial speed testing was done with the messaging application. Our benchmark involves sending 101 messages to a device, with the timer starting when message 0 arrives, and ending when message 100 arrives. We implemented this benchmark in both our text messaging applications - Intents and the AIDL library - as well as in the text messaging application built into the packet forwarding layer, completely free of our control structure. The difference was significant.
	1. *Speed*

The packet forwarding layer averaged 733ms for the benchmark, while our external application, communicating over Intents, averaged over 2600ms. It is clear that, while very convenient for small data transmissions, Intents are not going to be a wise implementation for applications with significant amounts of data to send. Our implementation using the AIDL library fared much better, though still rather slow compared to the raw connection, averaging near 1600ms.

* 1. *Application Implementation*

The interface structure between the application layer and the network operating system and network layers makes it much easier to develop external applications using the ad hoc network. It only takes 10-20 lines of code to use the AIDL interface library and less than 50 to implement Android’s intent system to setup and interact with the lower layers from an application. Implementing similar functionality directly using the Wi-Fi ad hoc service, even with a library to simplify its use, can take 100s or 1000s of lines of code because of the lack of abstraction.

1. Conclusion

This paper describes a way software-defined networking concepts can be applied to a mobile ad hoc network. Our implementation runs completely in the application layer, which results in reduced speed but makes the system significantly more adaptable. By separating the data and routing from the network management and control additional features can be added without the need to modify code for unrelated parts of the application. It also reduces the development time for applications that use the ad hoc network. The current speed using the fastest interface is still about twice as slow as directly interacting with the packet forwarding layer. This can probably be reduced significantly by improving the implementation.

Currently the system implements simple behavior for the network operating system and control program. In the future, the network operating system could be extended to use state of the art ad-hoc load balancing mechanisms and address assignment [8][9]. The control program could be extended to configure different routing protocols running simultaneously through the NOS; to manage security policies in the network; and to allow for more complex routing rules. The control program could also be modified to function as a service in the ad hoc network, migrating between nodes as needed using a system such as the one described in [10].

References

1. Open Networking Foundation. "Software-Defined Networking: The New Norm for Networks," April 13, 2012, available at https://www.opennetworking.org/images/stories/downloads/white-papers/wp-sdn-newnorm.pdf
2. The OpenFlow Switch Consortium, “OpenFlow Switch Specification Version 1.1.0”, 2011, available at http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf
3. Yi Shang; Wenjun Zeng; Ho, D.K.; Dan Wang; Qia Wang; Yue Wang; Tiancheng Zhuang; Lobzhanidze, A.; Liyang Rui; , "Nest: Networked smartphones for target localization," *Consumer Communications and Networking Conference (CCNC), 2012 IEEE* , vol., no., pp.732-736, 14-17 Jan. 2012.
4. Kapang Lego, Pranav Kumar Singh, Dipankar Sutradhar, "Comparative Study of Ad hoc Routing Protocol AODV, DSR and DSDV in Mobile Ad hoc NETwork," Indian Journal of Computer Science and Engineering, Vol. 1, No. 4, pp. 364-371., Dec. 2010.
5. C. Perkins, E. Belding-Royer, S. Das, "Ad Hoc On Demand Distance Vector (AODV) Routing," Request for comments, http://tools.ietf.org/html/rfc3561, July 2003
6. Android Interface Definition Language (AIDL), available at <http://developer.android.com/guide/components/aidl.html>
7. Intents and Intent Filters, available at <http://developer.android.com/guide/components/intents-filters.html>
8. Ganjali, Y.; Keshavarzian, A.; , "Load balancing in ad hoc networks: single-path routing vs. multi-path routing," *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies* , vol.2, no., pp. 1120- 1125 vol.2, 7-11 March 2004
9. Al-Shurman, M.; Al-Mistarihi, M.F.; Qudaimat, A.; , "Network Address Assignment In Mobile Ad-Hoc networks," *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2010 International Congress on* , vol., no., pp.287-294, 18-20 Oct. 2010
10. Hyun Jung La; Jeong Ran Jang; Soo Dong Kim; , "Self-Stabilizing Ecosystem for Service-Based Mobile Computing," *e-Business Engineering (ICEBE), 2011 IEEE 8th International Conference on* , vol., no., pp.193-200, 19-21 Oct. 2011