Flexible SDN control in tactical ad hoc networks

Konstantinos Poularakis a,*, Qiaofeng Qin a, Erich M. Nahum b, Miguel Rio c, Leandros Tassiulas a

aDepartment of Electrical Engineering and Institute for Network Science, Yale University, USA
bIBM Thomas J. Watson Research Center Yorktown Heights, NY USA
cDepartment of Electrical and Electronic Engineering, University College London, UK

A R T I C L E I N F O

Article history:
Received 10 June 2018
Revised 24 September 2018
Accepted 16 October 2018
Available online 18 October 2018

Keywords:
Software defined networks
Mobile ad hoc networks
Tactical operations

A B S T R A C T

Modern tactical operations have complex communication and computing requirements that cannot be supported by today’s mobile ad hoc networks. The emerging Software Defined Networking (SDN) paradigm has the potential to enable the redesign and successful deployment of these systems. An SDN-based approach, however, will also bring new challenges since the SDN architecture was not designed to accommodate the requirements of an ad hoc network environment. Unreliability and dynamism may fragment the tactical network making the centralized SDN controller unsafe. To address these issues, in this paper, we propose flexible protocols that split the control of the ad hoc network between the centralized SDN controller and the data plane nodes. The latter can dynamically decide whether to follow the controller instructions or adapt to network changes in a distributed manner. We implement a proof-of-concept prototype of a flexible SDN ad hoc system and perform experiments to measure its performance benefits over traditional OpenFlow. Going a step further, we study theoretical, yet practical, methods of managing the overheads of flexible control which are crucial for the successful large-scale deployment of these systems. Evaluations on a real tactical ad hoc network dataset demonstrate that up to 50% more packets can be routed to their destinations if flexible SDN control is enabled, while the extra overheads are one order of magnitude lower than the control message overheads of traditional OpenFlow systems.

© 2018 Elsevier B.V. All rights reserved.

1. Introduction

1.1. Motivation

Software Defined Networking (SDN) has emerged as a new and promising paradigm targeting to revolutionize the way communication networks are built and operate [1]. A key characteristic of SDN is the centralized network control which is facilitated by shifting the control functions from the data forwarding devices to a conceptually centralized network entity, the controller. The programmability provided by the controller has led to the development of efficient solutions to perform fine-grained context-aware resource management and introduce new services in the network. While most of the efforts of applying the principles of SDN have been in wired networks, for instance data centers and ISP networks [2], a more recent trend is to explore its extension to wireless and mobile networks. It is well understood by now that SDN can facilitate the fine-grained management of resources of the wireless infrastructure nodes such as radio access network switches [3] and base stations [4]. However, the application of SDN concepts in wireless networks with limited or poor infrastructure, such as tactical ad hoc networks, has only recently been attempted. The pioneer works in [5–9] have demonstrated the feasibility and performance benefits of applying SDN to a tactical ad hoc network. Compared to state-of-the-art distributed routing protocols (e.g., AODV and OLSR), an SDN-based approach for these networks can provide higher throughput, faster response to failures and prolong network lifetime.

At the same time, however, an SDN-based approach for tactical ad hoc networks will also bring new challenges. In fact, the SDN architecture was not designed to accommodate the requirements of an ad hoc network environment. In order for SDN to work properly, the continuous exchange of topology and routing table control messages between the controller and data plane (forwarding) nodes is needed [10]. In the tactical ad hoc environment, however, this operation is challenged by the low data rates of wireless channels and the inherent problem of unreliable connectivity. For example, in Fig. 1(a), the SDN controller that is located at a back-end server (e.g., the tactical command center) is unable to reach the handheld mobile devices of soldiers that move away from the ve-
vehicle used as a relay. Hence, it will be impossible to reconfigure them, resulting in outdated network policies.

A method that has been widely used to support network operations in areas without infrastructure is the deployment of Mobile Ad hoc Network (MANET) protocols, such as AODV and OLSR, to name two [11]. Although these protocols lack the centralized control and programmability of SDN, they are based on distributed routing and flooding schemes which have been shown to be quite robust and adaptive to network changes such as failures and additions of new links. For example, in Fig. 1(b), the soldier devices employing a MANET protocol disseminate to each other control messages such as list of neighbor nodes, power usage and topological position of the node. This way, they can discover alternative routing paths in a distributed manner and re-establish connectivity when network changes occur.

From the above discussion, the following question naturally arises: is it possible to design network control protocols that combine the benefits of the two paradigms; (i) programmability and centralized control of SDN and (ii) robustness and adaptivity of distributed MANET? In this paper, we answer this question affirmatively with flexible SDN, an architecture for the control of tactical ad hoc networks that borrows ideas from the existing MANET protocols to address potential limitations of SDN in this context.

1.2. Methodology and contributions

We begin by presenting the design of the proposed flexible SDN architecture. At the core of our design are simple control applications, referred to as local agents, that can be pre-stored and run locally by the data plane nodes. These applications enable nodes to disseminate to each other messages about changes in the state of links they detect, which can be exploited in routing packets in a distributed and coordinated manner. In a sense, these applications mimic basic operations of existing MANET protocols. A basic difference, however, is that they do not flood messages to all nodes in the network, but they target specific nodes. Thus, they are more lightweight than MANET protocols.

The purpose of the above message dissemination process, orchestrated by the local agents, is to enable the nodes to dynamically decide whether they will follow the SDN controller instructions (i.e., the forwarding rules installed by the controller in nodes’ OpenFlow tables), or they will unilaterally change their forwarding behavior. The SDN controller should be preferred when it is available and easy to access, to pick up the benefits of centralized network control and programmatically introduce new network-wide functionality. However, when the SDN controller becomes unavailable or costly to access, which are typical scenarios in tactical ad hoc networks, the nodes should rely on the control information of the messages disseminated to each other to realize basic, yet critical, network policies, such as re-establishing connectivity when topology changes occur.

A way to automate such a dynamic transition between centralized and distributed control is by using stateful extensions of OpenFlow protocol [12,13]. These extensions allow a data plane node to dynamically select different forwarding rules depending on the current network state (e.g., the set of links and routing paths currently available). Hence, by pre-storing a set of backup forwarding rules at the memory of the nodes, the latter can dynamically and in an automated way select between applying these rules (and therefore change their forwarding behavior unilaterally) or the rules installed by the SDN controller. This automation is important since it saves computation resources and time at the data plane nodes.

To demonstrate the feasibility of our flexible SDN design, we implement a proof-of-concept prototype using commercial off-the-shelf mobile devices (popular Android-based smartphones and laptops). These devices are programmed to run Open VSwitch SDN components [14] as well as the local agents we designed. We perform experiments to measure how quickly the local agents can react to wireless (WiFi) link failures and recoveries and compare with the conventional OpenFlow implementation in which a centralized controller handles all network events.

Next, we study theoretical, yet practical, methods of managing the overheads of flexible control which are crucial for the successful large-scale deployment of these systems. We model the problem of tuning the rate of control messages disseminated between the local agents aiming to support the local re-establishment of connectivity with the minimum possible message overhead. We show the high complexity of this problem and propose a greedy-based algorithm with provable approximation guarantees using the theory of submodular functions [15]. The overheads of the proposed algorithm are evaluated on a real-world tactical ad hoc network dataset containing fine-grained records of mobility and connectivity patterns between over a hundred military vehicles and soldiers [16]. The results demonstrate the efficacy of the proposed optimization approach.

The contributions of this work are summarized as follows:

- **Flexible SDN Control Architecture.** We propose and design flexible architectures that split the control of the tactical ad hoc network between the centralized SDN controller and the data plane nodes. The proposed architectures borrow ideas from existing MANET protocols and leverage latest stateful extensions of SDN protocols to accommodate the requirements of the ad hoc network environment.

- **Proof-of-Concept Prototype Implementation.** We implement a flexible SDN-enabled mobile ad hoc system and execute experiments to show the feasibility of our design. The experimentation results demonstrate that flexible SDN requires significantly lower delay for switching a routing path compared to a traditional OpenFlow system for a range of experiments.

- **Optimization Framework.** We present a system model and a framework to optimally tune the message dissemination between the data plane nodes. Our goal is to ensure that local re-establishment of connectivity between nodes can be achieved with the minimum possible message overhead. We show the high complexity of this problem and develop a practical approximation algorithm.

- **Dataset-driven Evaluation.** We evaluate the proposed algorithm using a real-world tactical ad hoc network dataset. We find that up to 50% more packets can be routed if flexible SDN control is enabled. The extra overheads of flexible SDN are one order of magnitude lower than the control message overheads of traditional OpenFlow.

The rest of the paper is organized as follows. The proposed flexible SDN architecture and its proof-of-concept prototype are presented in Sections 2 and 3 respectively. Section 4 develops a system model and an optimization framework for tuning the message...
dissemination between the nodes. Section 5 evaluates the overheads of flexible SDN on the real tactical ad hoc network dataset, while Section 6 reviews our contribution compared to the related works. We conclude our work in Section 7.

2. Flexible SDN architecture

In this section, we present the proposed flexible SDN architecture. We begin with an overview of the architecture and then provide details of each component.

2.1. Architecture overview

A high level description of the proposed architecture design is illustrated in Fig. 2. The traditional SDN architecture [1] is complemented with local agent application components running at the data plane node side. By virtue of the local agent operation, the data plane nodes are no longer passive, in the sense that they do not have to always apply the forwarding rules installed by the controller in their OpenFlow tables. This brings opportunities for data plane nodes to locally realize basic, yet critical, network policies, such as locally re-establishing connectivity when network changes occur, when the controller becomes unavailable or costly to access.

To support such critical network policies, the local agents incorporate mechanisms able to disseminate to each other control messages. These messages can be used to notify other agents about the state of the network such as the set of a node’s neighbor links that are currently available. In a sense, these mechanisms mimic basic operations of existing MANET protocols. To keep track of the currently available routing paths in the network, each node maintains a state table (similarly to OpenState [12] and FAST [13] extensions of OpenFlow). An entry of the state table can represent the availability or not of a routing path defined by a specific sequence of nodes. Depending on the current state table entries (i.e., the currently available routing paths), the local agent of the node decides whether to select a rule of the OpenFlow table (“primary rule”) to forward a packet (as in the traditional SDN) or unilaterally change node’s forwarding behavior.

To realize the latter operation, the local agent maintains also a local forwarding table which pre-fills with “backup” forwarding rules. These rules match the incoming data packets against both their headers (e.g., destination IP address) and state values (e.g., currently available routing paths). This way, when a local agent is notified about link changes by other agents, the respective state values are updated, and the backup rules are dynamically selected to forward incoming packets along available (backup) routing paths, instead of the primary paths.

2.2. Detection of neighbor nodes

The data plane nodes should keep track of their list of neighbor nodes (i.e., those nodes that are one hop away). A node can detect if another node is its neighbor by periodically broadcasting “heartbeat” packets. A neighbor node that receives such a packet will reply with the same packet. If no packets are received by a neighbor node for more than a given timeout, this node is no longer considered to be a neighbor. If later the node is again discovered, it will be added to the respective neighbor list.

This basic neighbor discovery operation is common in mobile ad hoc networks. When a traditional SDN-based approach (OpenFlow) for MANETs is adopted, nodes report their neighbor list information only to the central controller (cf. Fig. 1(a)). Alternatively, by adopting an existing (non-SDN) MANET protocol, the nodes flood their lists of neighbors across the whole network (cf. Fig. 1(b)). Flexible SDN provides a compromising solution, by disseminating the neighbor list information to specific nodes, as we explain below.

2.3. Control message dissemination between agents

The local agents of the nodes can disseminate control messages to notify each other about changes in the lists of neighbors of nodes. This will provide a form of consistency between the agents and allow them to coordinate their forwarding decisions. A straightforward solution would be to broadcast these messages to all the nodes in the network (similar to a MANET protocol). While this approach would ensure the strongest consistency that can be achieved, it would also induce significant message overheads. A more conservative approach, instead, where each agent picks a specific subset of the nodes to notify can provide the right tradeoff between consistency and overheads.

To realize such a constrained message dissemination between the local agents, we use a subscription mechanism where each local agent is notified only about the links between neighbor nodes it subscribed to. The dissemination of these messages will typically rely on multihop transmission. An agent that intends to share its neighbor links with another agent will have to transmit a message through a path of potentially many hops connecting the two agents. To increase robustness, the message can be sent through more than one network path.

2.4. Packet forwarding process

The packet forwarding process will depend on the state table, local forwarding table and OpenFlow forwarding table components. Consider for example the network in Fig. 3. The SDN controller has computed the primary routing path for a flow, which spans nodes 1, 2, and 3, and the respective forwarding rules have been stored in the OpenFlow tables of these nodes. When links on the primary path fail, the traffic of the flow should be dynamically rerouted to one of the two highlighted backup paths (if available). To support such flexible packet rerouting, the local agent of node 1 needs to be notified for the unavailability of the primary path, as well as the availability of the respective backup path and update the corresponding state values.

For example, assume that the primary path link (2,3) and the backup path link (4,3) fail (hence primary path and backup path 1 are unavailable), while the previously failed links (4,5) and (5,3) are added due to the mobility of node 5 (hence backup path 2 is available). The local agent of node 1 needs to be notified for all the above changes. If this agent is not notified for the failure of link (2,3), it will continue using the primary path, which is not available. Similarly, if the agent is not notified for the failure of link (4,3) it may incorrectly use the backup path (1,4,3) which is not available. Finally, if the agent is not notified for the addition of links (4,5) and (5,3), it cannot direct packets towards the backup path (1,4,5,3).

Given that the above notification messages will be delivered to the agent of node 1 by the other agents, we need to store the actual backup forwarding rules at the local tables of the nodes to
realize the dynamic rerouting towards the backup paths. To exemplify, we need to store one backup rule at each node for each backup path we want to support (two in our example). For node 1, these rules will match the packets of the flow against the state 'down' associated to the primary path (1,2,3) and 'up' associated to each of the backup paths, and forward them to the next hop node, i.e., node 4 (which happens to be the same for the two backup paths). Before forwarding, node 1 can tag¹ the packets with the sequence of nodes in the backup path, so as nodes 4 and 5 will know their next hop nodes. Hence, the backup rules installed at nodes 4 and 5 will match the packets of the flow with this tag and forward them to the next hop nodes. By placing one rule in a higher position in the local table than another, we give priority to one backup path over the other. In the example, backup path 1 is preferred (if available) from backup path 2.

2.5. Packet buffering and synchronization

Another challenge in realizing flexible packet forwarding is how to ensure consistency when the messages disseminated between the agents are delayed or lost. Consider for example the scenario in Fig. 3 where link (2,3) fails in the middle of the transmission of a data file over the primary path. Node 2 will send a notification message to node 1, so as the latter to stop forwarding data packets to it. However, the delivery of this message may be delayed, and hence a number of packets may be accumulated at the buffer of node 2 or dropped. Once node 1 receives the notification message will update its state values and forward the next packets of the file to node 4, using the first rule in its local table. Still, the packets that have been already pushed to node 2 will not receive a second chance to reach their destination.

To address this issue, node 1 can keep a copy of the packets it forwards in its buffer for a certain time window. If the routing path changes, node 1 can forward again the buffered packets to the next hop node in the new path. To avoid duplicate transmissions of packets, synchronization between node 1 and node 3 should be supported. To this end, node 3 can send acknowledgments of the packets it receives to notify node 1 that these packets are no longer needed to be buffered and hence can be safely dropped.

In the next section, we show that the above application components can be integrated into commercial off-the-shelf mobile devices, which makes our flexible architecture design directly applicable to the mobile market.

3. Proof-of-concept prototype

In this section, we implement a proof-of-concept prototype of the proposed flexible SDN architecture and perform experiments to measure its performance benefits over a traditional (non-flexible) OpenFlow system. The experimentation results demonstrate that flexible SDN requires significantly lower delay for switching a routing path compared to the fully-centralized OpenFlow system (20 vs 50 ms on average).²

3.1. Prototype set-up.

Our prototype testbed is built from modern off-the-shelf mobile devices. Namely, it contains four Nexus 4 Android-based smartphones and one Macbook Pro-laptop. We install popular SDN-related software in each device. The laptop runs the POX controller implementation [18], while the smartphones run the Open vSwitch (OvS) data plane implementation [14]. The Linux kernel of Android system makes this implementation possible. More specifically, we create a chroot container to hold OvS. The container also enables us to use some common desktop software in smartphones, including Wireshark [19], Iperf [20] and Iptables [21], which are widely adopted in network experiments and guarantee more reliable measurement results.

Besides of OvS, each smartphone runs a local agent. This contains an application that detects link changes, a state table and a local forwarding table. The application works in two modes, representing traditional and flexible SDN architectures. In the first mode, once a link change is detected, a special packet will be sent to the controller. The controller is appropriately programmed to calculate and send new forwarding rules containing alternative routing paths to the node. In the second mode, the application notifies the local agents of other smartphones, so that they will look up backup rules under the new state and apply them to OvS. In this way the link change is handled in a distributed manner.

With these devices, we form a small network verifying the example shown in Fig. 3. Two paths are available for sending a message from smartphone 1 to smartphone 3. Smartphone 2 offers a Wi-Fi hotspot to other devices including the laptop, which is regarded as the primary path. Moreover, smartphone 4 also connects with both smartphone 1 and 3 with Wi-Fi Direct links. We artificially put a bandwidth limitation on this link, so that it is treated as a backup path and is chosen only when the primary path is not available. Related settings are shown in Fig. 4. In Android system the two paths correspond to different network interfaces in each device and have different sets of IP and MAC addresses. The local agent uses OvS to take over the control of both interfaces. In order to choose a specific route, OvS forwards packets to the corresponding interface and conducts Network Address Translation (NAT), by properly modifying source and destination addresses in the IP and ARP packets according to the interface we choose.

3.2. Experimentation results

We experiment with the scenario that the link between the smartphone 2 and 3 fails. In this case, the flow between them is supposed to be migrated by the local agent to the WiFi Direct connection through smartphone 4. Fig. 5(a) depicts the transmission procedure of a UDP flow. After 30 s from establishing the flow, we

¹ Such function is supported by OpenFlow via VLAN or MPLS tags [17].
emulate a link failure on (2,3) by stopping the packet forwarding of smartphone 2. The local agent reacts quickly and migrates the flow to the backup path. Though tolerating a lower bandwidth provided by smartphone 4, the transmission is never interrupted. And after another 30 s we recover the link (2,3). In a similar manner the flow returns to the primary path for the higher bandwidth.

Having demonstrated the effectiveness of our local agent, we then measure the delay of switching the routing path in both modes. Two types of delays are considered. First, the delay comes from the transmission of control messages. In the traditional case, smartphone 2 reports to SDN controller about the failure, then the controller sends an OpenFlow message to smartphone 1. In the flexible case it is more straightforward, where smartphone 2 directly informs smartphone 1. Second, it also takes time to calculate/lookup the forwarding rules at the controller/local agent. Different from the transmission delay, due to superior processors and memories, the SDN controller held in the laptop is expected to be faster than the local agents held in smartphones. We measure both transmission and computation delays, and define the summation as total delay. Fig. 5(b) depicts CDF plots over 300 measurements. The relatively unstable wireless connections lead to high latencies. In traditional mode, the total delay has a high mean value and variance, sometimes going beyond 100 ms. In contrast, the flexible mode behaves a lot better.

Fig. 5(c) further shows the average value of both transmission and computation delays. We find that although it takes more time looking up the backup forwarding rules in smartphone than in laptop, the lower transmission delay of the flexible architecture compensates it and earns an overall better performance. Noticing that over the past few years smartphones have acquired far larger calculating capacity than the ones we use in the experiments (produced in 2012), there is still potential to achieve even better performance.

The above testbed and experiments showed the feasibility of the proposed flexible SDN systems and highlighted their performance benefits, in terms of switching path delay, compared to traditional OpenFlow. We note that the benefits will be even higher in more extreme scenarios where the controller becomes unavailable, and hence flexible SDN is the only method for switching paths. Due to the testbed’s small scale, however, the amount of control messages that needed to be disseminated between the local agents was small. In large networks with frequent failures, however, the overheads of these messages can be a major source of resource consumption. In the next section, we will focus on this issue and propose practical methods to manage the message overheads.

3.3. Main takeaways

Flexible SDN requires significantly lower delay for switching a routing path compared to traditional OpenFlow (2.5 times lower delay on average in the presented experiments). This delay comes mainly from the computation that takes place in the mobile devices.

4. Optimization framework

In this section, we propose a method for managing the overheads of the message dissemination between the local agents. We begin by presenting a tactical ad hoc network model and then formulate the optimization problem of minimizing the total amount of disseminated messages.

4.1. Model & problem formulation

We consider a general model of a tactical ad hoc network which consists of a possibly diverse set of nodes \( N \) such as soldiers with handheld devices, vehicles equipped with onboard wireless equipment, relay nodes, etc. The network is SDN-enabled in the sense that each node runs an SDN data plane implementation, like OvS, which allows it to receive forwarding rules from a centralized SDN controller. The latter may be located at a remote server (e.g., at the tactical command center) or even inside the ad hoc network. Moreover, each node runs a local agent application able to support flexible packet forwarding.

We study the network for a given period of time (e.g., a few hours). During this period, the topology may change due to several factors such as node mobility, battery drain and physical layer disturbances. The traffic between the nodes may change as well, as the operational and communication needs change. We consider a set of node pairs \( C \) that are expected to communicate to each other, e.g., the commander of a mission and an infantry squad member. For each node pair \( c \in C \), we denote by \( s_c \) and \( d_c \) the source and destination node respectively.

The SDN controller is aware of the initial network topology and uses this information to configure the nodes in the beginning of
the time period. That is, the controller computes for each node pair \( c \) a routing path \( p_c \) and installs the respective forwarding rules in the OpenFlow tables of the nodes along this path. We call this the primary routing path for node pair \( c \).

Since the topology dynamically changes, some of the primary routing paths may become temporarily or permanently unavailable. At the same time, different paths may become available for routing. The relative position in the topology of the nodes may even change completely as nodes move in the terrain. In general, the evolution of a mobile ad hoc network topology and hence the availability of paths between nodes are not known in advance and cannot be predicted with accuracy. However, tactical ad hoc networks have often a special topology structure and follow, to a large extent, a predetermined mobility pattern. For example, our analysis of a real tactical ad hoc network dataset in Section 5 revealed that soldiers often move in groups and are connected in specific ways depending on the group they belong.

Based on the regularity in mobility we empirically observed, we introduce the notation \( B_l \) to denote the set of paths that are likely to appear for a node pair \( c \). A path \( b \in B_l \) may be used (if available) for rerouting traffic when \( p_c \) breaks down. Each path \( b \) is essentially a sequence of nodes that happen to be connected in that order. Although in general the number of possible paths is exponential to the number of nodes, due to the aforementioned regularity this number can be quite limited in practice.

In a traditional (non-flexible) SDN system, each time a primary path breaks down, the controller is asked to reconfigure the nodes and reroute traffic towards an alternative path. In a flexible SDN system, however, the local agents may be able to handle such event locally. A sufficient condition for locally re-routting traffic from a primary path \( p_c \) to a path \( b \in B_l \) is the agent of node \( s_c \) to be aware of both the unavailability of primary path \( p_c \) and the availability of path \( b \). For example, in Fig. 3, it suffices for node 1 to be aware of the current state of links (1,2), (2,3), (1,4) and (4,3) to determine the availability of the respective paths and locally reroute traffic towards path \( b = (1,4,3) \). As we explained in Section 2, if node 1 possesses such information, it can update its local state values accordingly and pre-store a backup rule matching packets at these states and forwarding them to node 4.

We denote by \( L_4 \) the set of links the state of which needs to be known for locally rerouting traffic towards path \( b \in B_l \). In the above example, we have \( L_4 = \{1,2\}, \{2,3\}, \{1,4\}, \{4,3\} \). Then, we call path \( b \in B_l \) selectable if node \( s_c \) is aware of the current state of all the links in \( L_4 \). This depends on which notification messages are disseminated between the local agents. In our model, we use the binary optimization variable \( x_{nm}^l \in \{0,1\} \) to indicate if node \( n \in N \) notifies node \( m \in N \) for the state of its neighbor link \( l \in L \) \( x_{nm}^l = 1 \) or not \( x_{nm}^l = 0 \). Here, \( L \) denotes the set of possible links. These variables constitute the message dissemination policy:

\[
x = (x_{nm}^l \in \{0,1\} : n, m \in N, l \in L)
\]

(1)

Notification messages about the state of a link \( l \in L \) can be sent by the head node of this link, denoted by \( h_l \in N \). Therefore, we can define whether a path \( b \in B_l \) is selectable or not by the following function:

\[
y_b(x) = \prod_{l \in L_b} x_{h_l s_l}^l
\]

(2)

The above expression indicates that the path \( b \) will be selectable \( y_b(x) = 1 \) if the source node \( s_c \) is notified about the current state of all the links in the set \( L_b \). Otherwise, \( b \) will not be selectable \( y_b(x) = 0 \).

Intuitively, the more paths are selectable for a pair of nodes, the more likely becomes to re-establish connectivity between these nodes when network changes occur. However, in order to support a higher number of selectable paths, the agents will need to disseminate a higher number of notification messages. Hence, there is a tradeoff between the number of selectable paths and the overhead of message dissemination.

To optimize this tradeoff, we formulate the problem of minimizing the amount of disseminated messages under certain constraints on the set of selectable paths. Specifically, we require for each node pair at least a number \( K > 0 \) of paths to be selectable. The Message Dissemination Problem (MDP) is expressed as follows:

\[
\min \sum_{n, m, c \in N, n \neq m} \sum_{l \in L} x_{nm}^l
\]

(3)

subject to:

\[
\sum_{b \in B_c} y_b(x) \geq K, \forall c \in C
\]

(4)

\[
x_{nm}^l \in \{0,1\}, \forall n, m \in N, l \in L
\]

(5)

This is a discrete optimization problem and such type of problems are typically challenging to solve. The primary and backup paths for different node pairs may overlap to each other, which perplexes the problem. It is not difficult to show that MDP is NP-Hard since it is more general than the well-known NP-Hard set cover problem [22].

4.2. Approximation algorithms

In this subsection, we present an approximation algorithm for the MDP problem by expressing it as the minimization of a monotone submodular set function subject to covering constraints [15]. We begin with the definition of submodular set functions:

**Definition 1.** Let \( G \) be a finite set of elements, referred to as the ground set. A set function \( f : 2^G \rightarrow \mathbb{R} \) is submodular if for all subsets \( Y_1, Y_2 \subseteq G \) with \( Y_1 \subseteq Y_2 \) and every element \( g \in G \setminus Y_2 \) it holds:

\[
f(Y_1 \cup \{g\}) - f(Y_1) \geq f(Y_2 \cup \{g\}) - f(Y_2)
\]

(6)

In other words, the marginal value for adding an element \( g \) in a set decreases as the respective set expands.

In the MDP problem, we define the ground set \( G \) as follows:

\[
G = (g_b : b \in B, c \in C)
\]

(7)

An element \( g_b \), for some \( b \in B \) and \( c \in C \), indicates that the path \( b \) is selectable for re-establishing connectivity between the node pair \( c \). This is equivalent to disseminating messages to node \( s_c \) notifying it about the current state of all the links in \( L_b \). Similarly, a subset of elements \( Y \subseteq G \) corresponds to a message dissemination policy such that every path \( b \) for which \( g_b \in Y \) is made selectable. Therefore, there is a one-to-one correspondence between the elements in \( Y \) and the disseminated messages in \( x \).

Based on the above observation, the objective function of the MDP problem can be written as a set function:

\[
f(Y) = \sum_{n, m, c \in N, l \in L} \mathbb{1}_{\{g_b \in Y: c \in C, b \in B_c, c \in L_b, n = h_l, m = s_l\}}
\]

(8)

where \( \mathbb{1}_{\{1\}} \) is the indicator function, i.e., it is equal to one if the condition in the subscript is true, otherwise zero. In the above expression, for each triplet \( n, m, l \) the indicator function is equal to one if node \( n \) sends a message to node \( m \) for link \( l \) (i.e., if \( x_{nm}^l = 1 \)). This happens when there is at least one element \( g_b \) (for some path \( b \in B \)) in the set \( Y \) such that link \( l \) belongs to \( L_b \), \( m \) is the source node of pair \( c \) and \( n \) is the node that notifies \( m \) for that link.

We will prove the following lemma.

**Lemma 1.** The set function \( f(Y) \) is monotone submodular.

**Proof.** Monotonicity is obvious since any new path \( b \) selected in the set \( Y \) cannot decrease the number of disseminated messages.
To show submodularity, we use the fact that the sum of submodular functions is also submodular. Hence, it suffices to show that the following function is submodular for every pair of nodes \((n, m)\) and link \(l:\)

\[
f_{lm}^{f}(Y) = \mathbb{1}_{\{g_{n} \in Y : c \in c_{l}, \ n \in \mathcal{L}_{n}, \ m \in \mathcal{L}_{m}, \ n \neq m \}} \tag{9}
\]

We consider two sets \(Y \subseteq G\) and \(Y' \subseteq G\) such that \(Y' \subseteq Y\), and an element \(g_{n} \in G \setminus Y'\) to be added to both sets. This element corresponds to a path \(b^{*}\) for which \(b^{*} \in \mathcal{B}_{c}\) for some node pair \(c' \in \mathcal{C}\).

If \(l \notin \mathcal{L}_{n}\) or \(n \neq h_{i}\) or \(m \neq s_{i}\), then the dissemination of the state of link \(l\) from node \(n\) to node \(m\) is not needed to support rerouting over path \(b^{*}\). Hence, in all these cases the marginal cost is zero regardless of the set of selective paths: \(f_{lm}^{f}(Y' \cup \{g_{n}\}) - f_{lm}^{f}(Y') = f_{lm}^{f}(Y'' \cup \{g_{n}\}) - f_{lm}^{f}(Y'') = 0\).

Otherwise, the marginal cost will be either zero or one. As the set of selective paths \(Y\) expands the marginal cost can only reduce; become zero from one if the state of link \(l\) is already disseminated to node \(m\) by node \(n\) according to the existing set \(Y\). Hence, the function \(f_{lm}^{f}(Y)\) is submodular. \(\square\)

Based on the above lemma, the MDP problem can be expressed as the minimization of a monotone submodular function subject to \(K\) covering constraints. There exist various approximation algorithms for this type of problems [15]. Here, we focus on a simple, yet efficient, Greedy approximation algorithm. This algorithm starts with an empty solution \(Y = \emptyset\) and then iteratively augments elements to the current solution. At each iteration an element \(g_{n}\) is selected to be augmented. The selection satisfies the following two criteria:

(i) The path \(b\) must belong to a set \(\mathcal{B}_{c}\) for a node pair \(c \in \mathcal{C}\) for which the lower bound in constraint (4) has not been satisfied yet.

(ii) Among the paths that satisfy (i), the path \(b\) with the lowest marginal cost is selected, i.e., the path \(b\) with the minimum value of\( f(Y \cup \{g_{n}\}) - f(Y)\).

The procedure ends when all the lower bound constraints have been satisfied.

The following theorem summarizes the approximation guarantees of the described algorithm [15].

**Theorem 1.** The Greedy algorithm computes a \(\Delta\)-approximation solution, where \(\Delta\) is the maximum number of paths that can be used for re-establishing connectivity between a pair of nodes, i.e., \(\Delta = \max_{c \in \mathcal{C}} | \mathcal{B}_{c} |\).

Executing the Greedy algorithm will return the selectable (backup) routing paths \(Y\) from which we can find the message dissemination policy \(x\). In the next section, we show that Greedy requires the dissemination of significantly fewer messages than a baseline random scheme, demonstrating the efficacy of our optimization approach.

## 5. Evaluation results

In this section, we conduct a large-scale evaluation of the proposed flexible SDN architecture using a real-world tactical ad hoc network dataset [16]. This dataset contains fine-grained records of the mobility and connectivity patterns between over a hundred soldiers. As illustrated in Fig. 6, a troop of soldiers moves over a large area for around 2 h. During the movement, the relative locations of soldiers will change, which has impact on the signal strength of the mobile devices carried by soldiers. When two nodes (soldiers) are far away from each other, the signal weakens and nodes lose connectivity to each other. To show the impact of mobility, we plot the evolution of the network graph’s average degree, i.e., the number of neighbors of a node in Fig. 7(a), which is consistent with the movement tracks. For example, when the troop splits up, a node can connect to fewer neighbors. Overall, we observe high variance in node degrees across time, which indicates the high dynamics of link failures and additions in the ad hoc network.

The tactical ad hoc network has a special structure. Specifically, the troop contains four tank groups. Inside each group, a network is formed among 24 nodes in an ad hoc manner. Besides, in each group there are two nodes playing the role of gateways, which can communicate with gateways of other groups. Therefore when a node sends a message across groups, the routing path must contain gateway nodes of both the source and destination groups, otherwise it is meaningless. Paths with too many hops should also be ignored, because they lead to significant delays that should be avoided in the tactical scenario. We assume that the central SDN controller has the information of the role each node plays, so that it is able to exclude those meaningless paths. The rest of the paths containing less than 5 hops are regarded as \(B_{c}\). For every flow, there are several hundreds of such paths. Among them, a primary path and a certain number of backup paths will be selected using the Greedy algorithm we proposed in the last section. Without loss of generality, we create 10 traffic flows with randomly chosen source and destination nodes. Each flow has a unit rate.

We concentrate on verifying whether the selected backup paths can effectively route traffic when the primary path is unavailable. What is more, we are also interested in the overhead of control messages. Towards a more convincing evaluation, we not only make estimations based on simulation, but also conduct emulations with the help of Mininet [23]. This allows us to create multiple virtual switches representing every node in a single computer. Furthermore, the local agent in emulation is executable code, which can be directly deployed on real devices. During emulation, those codes work successfully in a large scale, and generate results consistent with the numerical simulations, demonstrating the reliability of the proposed flexible architecture.

In Fig. 7(b), we perform evaluations for different numbers of backup paths \((K\) from 1 to 15\), and measure the portion of traffic that can be routed without the participation of the central SDN controller. This is particularly important in tactical ad hoc networks where unreliability and dynamism may (temporarily) fragment the SDN controller from the data plane nodes. Here, \(K = 0\) means that the only opportunity for a packet to reach its destination is by following the available primary path (non-flexible SDN). We find that about one third of the packets can be routed this way. By applying \(K = 15\) backup paths, the routed traffic increases to above 50%. Therefore, compared with the primary path only case, more than 50% additional traffic is routed if flexible SDN is enabled. We note that the above numbers of backup paths are relatively small, and even commercial handheld devices should be capable
enough to support such numbers. Besides, consistency is shown in both the value and tendency between the simulation and emula
tion results. This implies that our model successfully captures the
traffic routing decisions of the local agents and it is capable of pro-
viding guidance in realistic cases.

Another metric we consider is the overhead of control messages
disseminated between the local agents. Specifically, we calculate the
total bit rate generated by the local agents for disseminating
messages to each other which is necessary to maintain the backup
paths chosen by the proposed Greedy algorithm. As a comparison,
we also calculate the total bit rate for another algorithm (Random)
that chooses the same number of backup paths randomly. 
Fig. 7(c) shows that the Greedy algorithm achieves a significantly
lower overhead than the Random scheme, demonstrating the effi-
cacy of our proposed optimization approach. Greedy can maintain
15 backup paths with merely one third of control messages com-
pared to the Random scheme.

To understand the magnitude of the overheads, we also con-
nect all the nodes to ONOS [24], a widely-used SDN controller that
is designed for scalability. We observe that the overheads of local
agents are one order of magnitude lower than the overheads of the
messages exchanged between the central controller and the data plane
nodes. Particularly, the operation of the controller generates control
messages at 4.32 Mbps rate on average, while the overheads of lo-
cal agent operation are on the scale of a few hundreds Kbps. There-
fore, the proposed flexible SDN architecture can efficiently adapt to
network changes without introducing much burden into the net-
work.

5.1 Main takeaways

Up to 50% more packets can be routed to their destinations if
flexible SDN control is enabled in the tactical ad hoc network. The
extra overheads paid are one order of magnitude lower than the
controller-node overheads in traditional SDN implementations.

6. Related work

Traditional SDN architectures rely on the centralized controller
to handle network changes and reconfigure the data plane nodes
accordingly. Recently, several mechanisms that handle network
decisions directly in the data plane (as opposed to the control plane)
have been developed, e.g., see the recent versions of OpenFlow in
[12,13] and current initiatives in [25]. The basic idea of these
works is to pre-compute and pre-store backup rules whose packet
forwarding behavior depends on the local state of the data plane
node [26]. This way if a link failure occurs, the adjacent to it node
can automatically redirect the packets to a backup path, without
enquiring the controller. However, these backup rules depend only
on the local state of the node, and hence they cannot handle mul-
tiple failures occurring in different parts of the network (which are
common scenarios in mobile ad hoc networks). Our proposed flexi-
ble SDN approach overcomes this limitation, by enabling the coor-
dination between nodes through a tunable message dissemination
application component.

Another approach that can be adopted to handle network
changes directly in the data plane is to use a legacy distributed
protocol “as-a-backup” to the SDN protocol. The distributed proto-
col can run in parallel with the SDN protocol to take control and re-
establish connectivity if needed. This approach has been shown to
be quite effective in OSPF-based wired networks of ISPs [27].
In the context of MANET, however, the distributed protocol (e.g.,
OLSR) may require long time to converge and induce significant
overheads for flooding discovery messages, especially in large net-
works. In contrast, our flexible SDN design requires minimal com-
putations by the data plane nodes and has tunable overheads,
which make it more suitable for lightweight mobile devices in ad
hoc networks.

Besides of the backup rule and distributed protocol as-a-backup
approaches, alternative architectures which combine centralized
distributed control have been proposed in [28–30]. These solu-
tions either direct packets through data plane nodes that have pre-
installed forwarding rules [28], or they employ rule cloning and
local actions at the data plane nodes [29], or they use local for-
warding tables to differentiate data traffic among nodes based on
their communication affinity [30]. All the above works, however,
target to improve the scalability of the control plane in large-scale
data center networks. In contrast, we focus on mobile ad hoc net-
work scenarios and develop lightweight protocols that are suitable
for resource constrained mobile devices, as shown by our proof-of-
concept prototype implementation.

For completeness, we note that another method that can be
used to improve the performance and reliability of the SDN ar-
chitecture is the deployment of many (physically distributed but
logically centralized) controllers across the network [31]. As we
showed in Fig. 5(b), however, the wireless links between the con-
troller and nodes can cause significantly slower reaction to failures
than our approach. Hence, the two approaches, controller place-
ment and flexible control, should be seen as complementary, not as
competitive.

7. Conclusion

In this paper, we made a first step towards bringing SDN into
tactical ad hoc networks. To this end, we designed and imple-
mented new, flexible control protocols which enable mobile nodes
to handle network changes in a distributed manner, thus remain operational even when the SDN controller is not available. The developed prototype is built from off-the-shelf (Android-based) mobile devices that are commonly used today, making our design directly applicable to the mobile market. Going a step further, we studied theoretical, yet practical, methods of managing the overheads of flexible SDN control which are crucial for the successful large-scale deployment of these systems. Evaluations on a real-world tactical ad hoc network dataset demonstrated that flexible SDN can be a viable solution approach to overcome the inherent problem of unreliable connectivity between the mobile nodes.

A topic of future work can be to extend the flexible SDN approach beyond routing applications, to flexibly control storage and computing resources distributed in the tactical ad hoc network. We expect that the technical details of the prototype implementation we provided will facilitate such research efforts in the future. 

Acknowledgment

This work was partly supported by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Numbers W911NF-16-3-0001, and the ONR awards N00014-14-1-2190 and N00014-17-1-2427. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. K. Poularakis acknowledges the Bodoskki Foundation, Greece, for a postdoctoral fellowship.

References


Konstantinos Poularakis obtained the Diploma, and the M.S. and Ph.D. degrees in Electrical Engineering from the University of Thessaly, Greece, in 2011, 2013 and 2015 respectively. In 2014, he was a Research Intern with Technicalcolor Research, Paris. Currently, he is a Post-Doctoral researcher at Yale University. His research interests lie in the area of network optimization with emphasis on emerging architectures such as software defined networks and content distribution wireless networks. He was the recipient of several awards and scholarships during his studies, from sources including the Greek State Scholarships foundation (2011), the Center for Research and Technology Hellas (2012), the Alexander S. Onassis Public Benefit Foundation (2014) and the Bodoskki Foundation (2016). He also received the Best Paper Award at the IEEE Infocom 2017.

Qiaofeng Qin received his B.S. degree in electrical engineering from Peking University, China, in 2015. He is currently a PhD student at the Department of Electrical Engineering at Yale University. His research focuses on the area of communication networks, including Software Defined Networking and Internet of Things.

Erich M. Nahum received the Ph.D degree in computer science from the University of Massachusetts, Amherst, in 1996. He is a Research Staff Member with the IBM T. J. Watson Research Center, Hawthorne, NY. He is interested in all aspects of performance in experimental networked systems.
Miguel Rio is currently a Professor with the Department of Electronic and Electrical Engineering, University College London, where he researches and lectures on Internet technologies. His current research interests include real-time overlay streaming, network support for interactive applications, and quality of service routing and network monitoring and measurement.

Leandros Tassiulas is the John C. Malone Professor of Electrical Engineering at Yale University. His research interests are in the field of computer and communication networks with emphasis on fundamental mathematical models and algorithms of complex networks, architectures and protocols of wireless systems, sensor networks, novel internet architectures and experimental platforms for network research. His most notable contributions include the max-weight scheduling algorithm and the back-pressure network control policy, opportunistic scheduling in wireless, the maximum lifetime approach for wireless network energy management, and the consideration of joint access control and antenna transmission management in multiple antenna wireless systems. Dr. Tassiulas is a Fellow of IEEE (2007). His research has been recognized by several awards including the IEEE Koji Kobayashi computer and communications award, the inaugural INFOCOM 2007 Achievement Award "for fundamental contributions to resource allocation in communication networks," the INFOCOM 1994 best paper award, a National Science Foundation (NSF) Research Initiation Award (1992), an NSF CAREER Award (1995), an Office of Naval Research Young Investigator Award (1997) and a Bodossaki Foundation award (1999). He holds a Ph.D. in Electrical Engineering from the University of Maryland, College Park (1991). He has held faculty positions at Polytechnic University, New York, University of Maryland, College Park, and University of Thessaly, Greece.