Flexible SDN Control in Tactical Ad Hoc Networks

Konstantinos Poularakis[‡], Qiaofeng Qin[‡], Erich Nahum[◇], Miguel Rio[*], Leandros Tassiulas[‡]

[‡]Department of Electrical Engineering and Institute for Network Science, Yale University, USA
[◇]IBM T. J. Watson Research Center, USA
[*]Department of Electrical and Electronic Engineering, University College London, UK

***Abstract*—Modern tactical operations have complex communication and computing requirements, often involving different coalition teams, that cannot be supported by today's mobile ad hoc networks. To this end, 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 environment. Indeed, 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 distributed manner. We implement a proof-of-concept prototype of a flexible SDN ad hoc system and perform experiments to measure its performance and overheads. Going a step further, we study theoretical, yet practical, methods of managing the overheads of flexible control which are crucial for the success of these systems.**

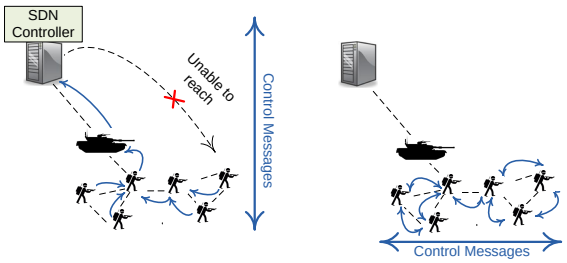
I. Introduction

A. Motivation

Software Defined Networking (SDN) has emerged as a new paradigm promising to revolutionize the way networks are built and operate [1]. A key characteristic of SDN is the *centralized network management* 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 manage and configure the network "on the fly" as well as dynamically introduce new services to end-users.

While most of the efforts of applying the principles of SDN have been in wired networks, such as data centers [2] and ISP networks [3], a more recent trend is to explore its extension to wireless networks. It is well understood by now that SDN can enhance the resource management of the wireless infrastructure nodes such as access switches [4] and base stations [5]. However, the application of SDN concepts in *wireless networks without infrastructure*, such as tactical ad hoc networks, remains unexplored.

Despite its inherent benefits of programmability and centralized control, an SDN-based approach for tactical ad hoc networks will also bring new challenges [6], [7], [8]. In fact, the SDN architecture was not designed to accommodate the requirements of an ad hoc environment. In order for SDN to work properly, the continuous exchange of topology and routing table information between the controller and data plane nodes is needed [9]. In the tactical ad hoc environment, however, this operation is challenged by the low data rates of channels and the inherent problem of unreliable connectivity. For example, in Figure 1(a), the SDN controller that is located at a remote server (e.g., the command center) is unable to reach the soldiers that move away from the vehicle used as an access point. Hence, it will be impossible to reconfigure them, resulting in outdated network policies.



(a) SDN-based centralized control. (b) Distributed control.

Fig. 1: SDN-based vs distributed approach.

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 [10]. 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 dynamics such as link failures and additions. For example, in Figure 1(b), the soldiers employing a MANET protocol maintain and disseminate information 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 to re-establish connectivity and adapt when network changes occur.

In this work, we borrow ideas from the existing distributed MANET protocols to address the limitations of the SDN architecture and make it more suitable to the tactical ad hoc network environment. Our goal is to design *flexible* control protocols that combine the benefits of the two paradigms; (i) centralized control and programmability of SDN and (ii) robustness and adaptivity of distributed MANET.

To exemplify, we design simple *control functions* that can be pre-stored and run *locally* by the data plane nodes, without the involvement of the centralized SDN controller. These functions allow the nodes to detect failures and additions of new links to neighbor nodes, maintain this information in their local memories and disseminate this information to other nodes,

even if the latter are located several hops away. In a sense, these functions *mimic basic operations of MANET protocols*. A basic difference, however, is that the designed functions do not flood messages to all nodes in the network, but only to specific nodes. Thus, they are more lightweight than MANET protocols.

The purpose of the above control functions 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 actions. The SDN controller should be preferred when it is available and easy to access, to pick up the benefits of *global network view* and implement sophisticated network policies. 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 their own *local views of the network* to realize basic, yet critical, network policies, such as re-establishing routing paths when network failures happen.

A way to realize such a dynamic transition between centralized and local control paradigms is by using the concept of *stateful SDN data plane*. Namely, Openstate [11] and FAST [12] are two stateful extensions of OpenFlow protocol that allow a programmer to specify how the data plane nodes should dynamically apply different forwarding rules depending on changes in network state (e.g., depending on the set of links observed to have been added or failed). Hence, by pre-storing a set of *backup forwarding rules* at the memory of the nodes, the latter can dynamically and in an automatic way select between applying these rules (and therefore change their forwarding behavior unilaterally) or the rules installed by the SDN controller.

B. Methodology and Contributions

We first describe the *design* of the proposed flexible SDN architecture. The core of this design is to grant additional capabilities to the data plane nodes by installing local control functions able to detect network changes, coordinate themselves, pre-store backup rules and effectively re-route packets. We describe the mechanism that enables each of these operations. Then, we provide a *proof-of-concept prototype* of the flexible architecture built from off-the-shelf wireless devices (commonly used smartphones and laptops). These devices are programmed to run the Open VSwitch (OVS) data plane implementation [13] as well as the local functions we described above. We perform experiments to measure how quickly the local functions of the devices can react to wireless link failures and compare with the conventional OpenFlow implementation in which a centralized controller manages all failures.

Having shown the feasibility of the proposed flexible SDN systems, 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. Namely, we model the problems of tuning the dissemination rate of control messages between the nodes and designing their packet forwarding logic aiming to support the local re-establishment of connectivity between critical pairs of nodes with the minimum message overhead. We show the high complexity of these problems and propose algorithms with provable approximation guarantees using the theory of submodular functions [14]. The overheads of the proposed algorithms are evaluated on a real wireless network topology.

The contributions of this work are summarized as follows:

- *Flexible SDN Control in Tactical Ad Hoc Networks*. We propose and design flexible protocols that split the control of the ad hoc network between the centralized SDN controller and the data plane nodes.
- *Proof-of-Concept Prototype Implementation.* We implement a flexible SDN-enabled mobile ad hoc system and execute experiments to highlight its benefits. We find that flexible SDN can provide a multi-fold reduction in failure reaction time compared to the conventional (fully-centralized) OpenFlow system for a range of experiments.
- *Optimization Framework*. We present a framework to optimally tune the message dissemination between the data plane nodes and store the dynamically-selectable backup forwarding rules. Our goal is to ensure that local re-establishment of connectivity between critical pairs of nodes can be achieved with the minimum total message overhead. We show the high complexity of these problems and develop approximation algorithms.
- *Trace-driven Evaluation.* We evaluate the proposed algorithms on a real wireless network topology. We find that a dissemination rate of no more than a few messages per minute suffices to support a link failure rate of more than 10%.

The rest of the paper is organized as follows. In Section II, we describe the proposed flexible SDN design and provide a proof-of-concept prototype. We show how to tune the overhead of message dissemination between the nodes and store the back-up forwarding rules in Sections III and and IV respectively. Section V evaluates the overheads of flexible SDN, while Section VI reviews our contribution compared to the related works. We conclude our work in Section VII.

II. Flexible SDN: Design & Proof-of-concept

In this section, we describe the design of the proposed flexible SDN architecture. As a proof-of-concept, we implement a real flexible SDN-enabled mobile ad hoc system and execute experiments to highlight its benefits.

A. Architecture design

In this subsection, we discuss the aspects of the proposed flexible SDN design: the architecture, the network change detection and control message dissemination mechanisms and the packet forwarding process.

Architecture overview: The proposed architecture design is depicted in Figure 2. We employ a *hybrid control model* where the data packets can be forwarded either according to flow rules provided by the SDN controller or by control mechanisms locally run by the data plane nodes. To realize local operation, we endow data plane nodes with additional capabilities. Specifically, we install and run in each node a

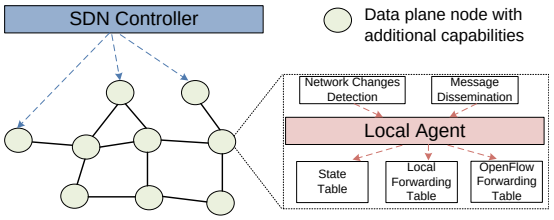


Fig. 2: Overview of flexible SDN control architecture.

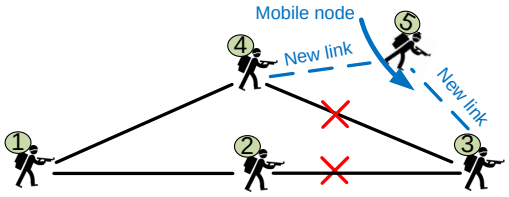


Fig. 3: Example of flexible packet forwarding.

local control function, which we call *local agent*, able to unilaterally alter node's forwarding behavior. Hence, the data plane nodes are not passive any more, in the sense that they do not have to always follow the flow rules installed by the controller. This brings opportunities for data plane nodes to locally realize basic, yet critical, network policies, such as re-establishing routing paths when network failures happen, when the controller becomes unavailable or costly to access.

To realize such critical network policies, the local agents run mechanisms to *detect network changes* and *disseminate messages to notify each other* about the detected events. In a sense, these mechanisms mimic basic operations of MANET protocols. To keep track of the available routing paths in the network, each node also maintains a *state table* (similarly to OpenState [11] and FAST [12] extensions of OpenFlow). An entry of the state table can represent the set of network links that are currently detected as failed or new links that have been added. Depending on the current state table entries, the local agent of the node can decide to select a rule of the OpenFlow table ("primary rule") to forward a packet or unilaterally change its forwarding action.

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., set of failed or newly added links). Hence, when link changes are detected, the backup rules can be selected dynamically and in an automatic way to forward packets to available (backup) routing paths, instead of the primary paths.

Detection of network changes: The local agents of the nodes are responsible for detecting the network changes. An agent of a node can detect the failure or addition of the adjacent links by exchanging "heartbeat" packets with the neighbor nodes. A node that receives such a packet will reply with the same packet. If no packets are received for more than a given timeout, the respective link is declared 'down' and the state associated to this link is updated accordingly (from 'up' to 'down'). If later the link is recovered, the state will change to 'up' again.

Control message dissemination between agents: Besides of the above basic operation of exchange of heartbeat messages, the agent of a node can also send messages to the agents of other nodes to notify them about detected changes to the state values. This provides a form of *consistency* between the agents and allows them to coordinate their forwarding decisions. A straightforward solution would be to broadcast the notification 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 agents, we can use a *subscription* mechanism where each agent will be notified only for state changes to the links it subscribed to. In general, the dissemination of notification messages will rely on multihop forwarding. An agent that intends to share its state with another agent will have to transmit a message through a path of potentially many hops connecting the two agents.

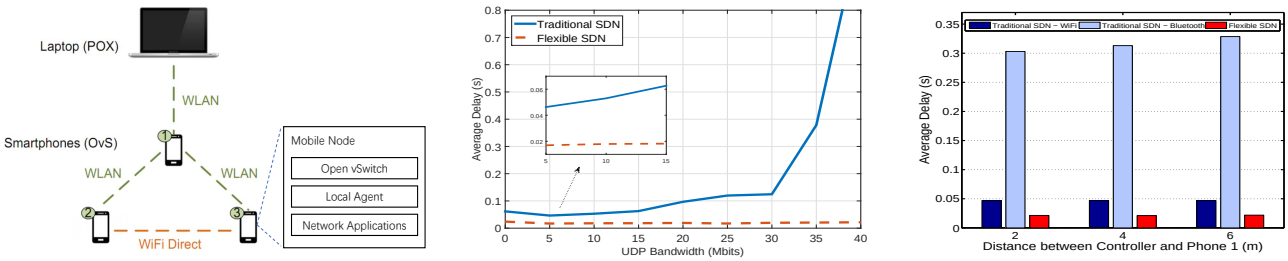
Packet forwarding: To understand the packet forwarding process in flexible SDN consider the example in Figure 3. The centralized controller has computed the primary routing path for a flow, which spans nodes 1, 2 and 3, and the respective forwarding rules have been stored in the OpenFlow tables of these nodes. Later, however, the intermediate link (2,3) and the link (4,3) fail while the new links (4,5) and (5,3) are added (due to the mobility pattern of node 5). The agent of each node is able to detect the failure and addition of its adjacent links only. To support local re-routing of the flow towards the available backup path (1,4,5,3), the agent of node 1 needs to be notified for the above link 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 path (1,4,5,3).

Given that the above notification messages will be delivered to node 1, we need to store the actual backup forwarding rules at the local tables of the nodes to realize such dynamic re-routing. In this example, it suffices to store a backup rule at each of the nodes 1, 4, 5 and 3. For node 1, this will match the packets of the flow against the state 'down' associated to the failed link (2,3) and 'up' associated to the links on the backup path, i.e., links (1,4), (4,5), (5,3), and forward them to the next hop node, i.e., node 4. Before forwarding, node 2 can *tag*[1] the packets with the sequence of nodes in the backup path, so as nodes 4 and 5 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.

B. Proof-of-concept prototype

In this subsection, we provide a proof-of-concept prototype of the above flexible SDN architecture and perform experi-

[1]Such function is supported by OpenFlow via VLAN or MPLS tags [15].



(a) Proof-of-concept prototype. (b) Average delay for updating routing. (c) Impact of technology and distance.

Fig. 4: (a) The prototype involving three OvS-enabled smartphones and a laptop running the POX controller. (b) Average delay under different extents of congestion. (c) Average delay for different wireless technologies and distance to controller.

ments.

Prototype set-up. Our prototype testbed is built from modern off-the-shelf mobile devices. Namely, it contains three Nexus 4 Android smartphones and one Macbook Pro laptop. We install popular SDN-related software in each device. The laptop runs the POX controller implementation [16], while the smartphones run the Open VSwitch (OvS) data plane implementation [13].

Besides of OvS, each smartphone runs a local agent. This contains an application that detects link failures, 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 failure 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 directly connects with the state table of local agent, then looks up backup rules under the new state and applies them in OvS, so that the link failure is handled locally.

The wireless ad hoc mode is not permitted by the devices we use. However, multiple network interfaces and protocols are supported (e.g. 3G, WiFi hotspot, WiFi Direct, Bluetooth), which makes it possible to form a wireless ad hoc network as it is depicted in Figure 4(a). Namely, we use one of the smartphones as a WiFi access point (hotspot) to connect with the laptop and the other two smartphones, while we also connect the latter through the WiFi direct protocol. Therefore, two routes exist between any pair of smartphones. 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, with properly modifying source and destination addresses in the IP and ARP packets according to the interface we choose.

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 to the WiFi hotspot connection through smartphone 1. We run the failure detection application in both traditional and flexible modes. In addition, another UDP flow is created between the laptop and the smartphone 1 by iperf, of which the bandwidth is controlled to show the impact of congestion in the network.

We measure the delay between the detection of link failure and the update of the forwarding rules in response to this failure. We do this separately for each of the two modes. We allocate different values to the bandwidth of the UDP flow, and take 200 measurements for each value. The results in Figure 4(b) show that *the average delay in flexible mode is smaller than in the traditional mode*, because it saves time from communicating with the controller, despite the overhead of computing with limited capacity of the smartphone. *When the congestion of the link to controller is heavy, the advantage of flexible mode is more notable (multi-fold reduction in delay is reported).*

We next explore how the results change when the WiFi link between laptop and smartphone 1 is replaced by a Bluetooth link (Figure 4(c)). The delay in the flexible mode is independent of the wireless technology, but in the traditional mode increases by 6 times because of the lower rate of Bluetooth. Increasing the distance to controller (2m per link in our initial setup) further increases the delay in traditional mode.

The above testbed and experiments showed the feasibility of the proposed flexible SDN systems and highlighted their benefits compared to conventional OpenFlow. However, due to the testbed's small scale, the notification message dissemination and backup rule storage processes were straightforward; the smartphones did not need to disseminate any message and only one backup routing path spanning three nodes existed. In large networks with frequent failures, however, the overheads of these processes can be significant. In the next section, we propose a method to manage the message dissemination overheads. Following that, in Section IV, we describe how to store the backup rules for a given message dissemination strategy, and discuss practical methods to ensure consistent packet forwarding.

III. Message Dissemination between Agents

In this section, we propose a method for managing the overheads of message dissemination between the agents. We begin by presenting a model and defining the problem of tuning the message dissemination between the agents.

A. Model

We consider a general model representing a tactical ad hoc network that contains a possibly diverse set of nodes such as handheld devices, vehicles equipped with onboard wireless devices, relays, 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. Moreover, each node runs a local agent able to support flexible packet forwarding.

The topology of the network may change over time due to several factors such as node mobility and physical layer disturbances. Without loss of generality, we assume that the set of nodes remains the same during a period of study and focus on the link changes, i.e, failures and additions of links. We denote the set of nodes by $\mathcal{N}$, where $N = |\mathcal{N}|$.

Due to the link failures, some of the paths used to route traffic may become unavailable, and hence the connectivity between some pairs of nodes may be lost. The tactical network operator will need to quickly re-establish connectivity between these node pairs and re-route traffic using paths that are currently available. Depending on the tactical operations, this may be more urgent for certain "critical" pairs of nodes than others, e.g., for re-establishing connectivity between the commander of a mission and an infantry squad.

To model such cases, we denote by $\mathcal{C}$ a set of critical pairs of nodes, where $C = |\mathcal{C}|$. For each critical pair $c \in \mathcal{C}$, we denote by s_c and d_c the two critical nodes and by p_c the routing path used to connect them, also called the *primary routing path*. We also denote by $\mathcal{B}_c$ a set of *backup* paths that may be used (if available) for rerouting traffic when p_c breaks down.

A sufficient condition for locally re-routing traffic from a primary path p_c to a backup path $b \in \mathcal{B}_c$ is that the agent of the node s_c has knowledge of the link failures on the primary path p_c and the availability of links on the backup path b. We use the notation $\mathcal{L}_b$ to denote this set of links. From now on, we call a backup path $b \in \mathcal{B}_c$ as *selectable* for re-establishing connectivity between the critical pair of nodes c if node s_c is aware of the changes in the state of all the links in $\mathcal{L}_b$.

For example, in Figure 3, it suffices for node 1 to have knowledge of the failures of links (1,2) and (2,3) and the availability of links (1,4), (4,5) and (5,3) to re-route traffic towards backup path $b = (1, 4, 5, 3)$. Hence, it should be $\mathcal{L}_b = \{(1,2),\ (2,3),\ (1,4),\ (4,5),\ (5,3)\}$. Indeed, if node 1 possesses such information, it can update its local states accordingly and pre-store a backup rule matching at these states and forwarding packets to node 4. Before forwarding, node 1 can tag the packets with the sequence of nodes in the backup path, so as nodes 4, 5 and 3 become aware of their next hop nodes. We explain the backup rule storage process in detail in the next section.

Based on the above discussion, we can model which backup paths are selectable and which not as a function of the notification messages disseminated between the agents. We start by introducing the binary optimization variable $x^l_{nm} \in \{0, 1\}$ that indicates whether the agent of node $n \in \mathcal{N}$ notifies node $m \in \mathcal{N}$ for the changes in the state of the link $l \in \mathcal{L}$ ($x^l_{nm} = 1$) or not ($x^l_{nm} = 0$). Here, $\mathcal{L}$ denotes the set of possible links. These variables constitute the *message dissemination policy*:

$$\boldsymbol{x} = (x^l_{nm} \in \{0, 1\} \ : n, m \in \mathcal{N}, l \in \mathcal{L}). \tag{1}$$

The network links can be directed and asymmetric in general, meaning that a link pointing from a node a (head) to another node b (tail) does not imply a link from b to a. Therefore, only the agent of the head node of a link can send notification messages about its state. We use the notation h_l to denote the head node of link l. We also use the notation $y_b(\boldsymbol{x}) \in \{0, 1\}$ to indicate whether a backup path $b \in \mathcal{B}_c$ is selectable ($y_b(\boldsymbol{x}) = 1$) or not ($y_b(\boldsymbol{x}) = 0$), which is given by the following expression:

$$y_b(\boldsymbol{x}) = \prod_{l \in \mathcal{L}_b} x^l_{h_l s_c}. \tag{2}$$

The above expression indicates that the backup path b will be selectable if the first node of the critical pair, s_c, is notified about state changes of all links in the set $\mathcal{L}_b$.

Intuitively, the more backup paths are selectable for a critical 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 backup paths, the agents will need to disseminate a higher number of notification messages. Hence, there should be a balance between the overheads of message dissemination and the number of selectable backup paths.

The network operator may be concerned about meeting a specific reliability goal when designing the message dissemination policy. For example, it may require that at least a given number $K \in \mathcal{Z}^+$ of backup paths for each critical pair of nodes are made selectable. In this case, the objective should be to minimize the total message overhead while ensuring that the above lower bound on the number of selectable backup paths is met. We call this the Message Dissemination Problem (MDP):

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \textstyle\sum_{n,m \in \mathcal{N}, n \neq m} \sum_{l \in \mathcal{L}} x^l_{nm} \\ s.t. \quad & \sum_{b \in \mathcal{B}_c} y_b(\boldsymbol{x}) \geq K, \ \forall c \in \mathcal{C} \ , \qquad (3) \\ & x^l_{nm} \in \{0, 1\}, \forall n, m \in \mathcal{N}, l \in \mathcal{L} \ . \qquad (4) \end{aligned}$$

This is a discrete optimization problem, and such problems are typically challenging to solve. It is not difficult to show that MDP is more general than the set cover problem, and hence it is NP-Hard.

B. Approximation Algorithms

We now derive a class of approximation algorithms by expressing the MDP problem as the minimization of a submodular set function subject to covering constraints [14]. We begin with the definition of submodular set functions:

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

$$f(Y^1 \cup \{g\}) - f(Y^1) \geq f(Y^2 \cup \{g\}) - f(Y^2). \tag{5}$$

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

In our problem, we define the ground set $\mathcal{G}$ as follows:

$$\mathcal{G} = (g_b : b \in \mathcal{B}_c, c \in \mathcal{C}), \tag{6}$$

where an element g_b, for some $b \in \mathcal{B}_c$, indicates that the backup path b is selectable for re-establishing connectivity between the critical pair of nodes c. This is equivalent to notifying node s_c about the state changes of all the links in $\mathcal{L}_b$. Similarly, a subset of elements $Y \subseteq \mathcal{G}$ corresponds to a message dissemination policy such that every backup path b for which $g_b \in Y$ is selectable. Hence, *there is an one-to-one correspondence between the elements in Y and the disseminated messages in $\boldsymbol{x}$.*

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

$$f(Y) = \sum_{\substack{n,m \in \mathcal{N} \\ n \neq m}} \sum_{l \in \mathcal{L}} \mathbb{1}_{\{|g_b \in Y \,:\, c \in \mathcal{C},\, b \in \mathcal{B}_c,\, l \in \mathcal{L}_b,\, n = h_l,\, m = s_c| \geq 1\}}\,, \tag{7}$$

where $\mathbb{1}_{\{\}}$ 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 $x_{nml} = 1$). This happens when there is at least one element g_b, for some backup path $b \in \mathcal{B}_c$, in the set Y such that link l belongs to $\mathcal{L}_b$, m is the first of the nodes in the critical pair c and n is the node that notifies m for that link.

Then, it is important to show 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:

$$f^l_{nm}(Y) = \mathbb{1}_{\{|g_b \in Y \,:\, c \in \mathcal{C},\, b \in \mathcal{B}_c,\, l \in \mathcal{L}_b,\, n = h_l,\, m = s_c| \geq 1\}}\,. \tag{8}$$

We consider two sets $Y^1 \subseteq G$ and $Y^2 \subseteq G$, where $Y^1 \subseteq Y^2$, and an element $g_{b^*} \in \mathcal{G} \setminus Y^2$ to be added to both sets. This element corresponds to a backup path b^*, where $b^* \in c^*$ for some critical pair of nodes c^*.

If $l \notin \mathcal{L}_{b^*}$ or $n \neq h_l$ or $m \neq s_c$, then the dissemination of the state of link l from node n to node m is not needed to support rerouting over backup path b^*. Hence, in all these cases the marginal cost is zero regardless of the set of selectable backup paths: $f^l_{nm}(Y^1 \cup \{g_{b^*}\}) - f^l_{nm}(Y^1) = f^l_{nm}(Y^2 \cup \{g_{b^*}\}) - f^l_{nm}(Y^2) = 0$.

Otherwise, the marginal cost may be zero or one. Clearly, as the set of selectable backup paths expands the marginal cost can only reduce; become zero if the state of link l is already disseminated to node m by node n. Hence, the function $f^l_{nm}(Y)$ is submodular. ■

Based on the above lemma, the MDP problem can be expressed as the minimization of a submodular function subject to C covering constraints. There exist various approximation algorithms for this type of problems [14]. Here, we focus on a simple greedy algorithm. This algorithm starts with an empty solution, $Y = \emptyset$, and iteratively adds an element g_b in the solution. The backup path b must belong to a set $\mathcal{B}_c$ for a critical pair of nodes c for which the lower bound of K in constraint (3) has not been reached yet. Among the available backup paths, the backup path b that results the minimum marginal cost is selected, i.e., the backup path b with the minimum value of $f(Y \cup \{g_b\}) - f(Y)$. The procedure ends when all the lower bound constraints have been satisfied.

The following theorem summarizes the approximation guarantees of the above algorithm [14].

Theorem 1: Greedy algorithm achieves a Δ-approximation solution, where Δ is the maximum number of backup 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 above algorithm will return the selectable backup routing paths Y from which we can find the message dissemination policy $\boldsymbol{x}$. We then need to install the actual backup forwarding rules which agents will use to reroute traffic to the backup paths. We describe how exactly to store these backup rules, along with methods to ensure consistent packet forwarding, in the next section.

IV. Backup Rule Storage & Consistency

In this section, we elaborate on the backup rule storage process necessary to enable the rerouting of flows to the backup paths computed in the previous section. Following that, we describe how to avoid inconsistency when the messages disseminated between the agents are delayed or lost.

A. Backup Rule Storage

We will compute the backup forwarding rules for each critical pair of nodes independently from the other pairs. Consider a specific critical pair $c \in \mathcal{C}$. The greedy algorithm proposed in the previous section returns a set of backup paths that can be used to reroute the traffic between this pair when failures occur on the primary path p_c. For a backup path $b \in \mathcal{B}_c$ returned by this algorithm ($g_b \in Y$), node s_c is notified about the states of the links on the backup path b and the primary path p_c, as we explained in the previous section. Hence, the node s_c can maintain a state value ('up' or 'down') for each of these links which can use to match the incoming packets. This way, the node s_c can reroute the packets to the next hop node on the backup path b.

Consider for example the critical pair of nodes (1,3) in Figure 5. The traffic between these nodes can be rerouted to one of the two highlighted backup paths when links on the primary path have failed. Node 1 is notified about these link failures and set its local state for these links from 'up' to 'down'. Besides, node 1 has installed six backup forwarding rules matching the incoming packets (e.g., those having source address ('Src') node 1 and destination address ('Dst') node 3) when the current local state is 'down' for at least one of the links on the primary path and all the links are 'up' in the backup paths. These rules forward the packets to the next hop node, which in this example is common for the two backup paths (node 4).

Node 1 can tag the packets before forwarding to notify the intermediate nodes on the backup path about this rerouting. The intermediate nodes will forward the incoming packets to the next hop node on the backup path, provided that a backup forwarding rule matching the packets with the respective tag is stored. For example, a backup rule of this type is stored in

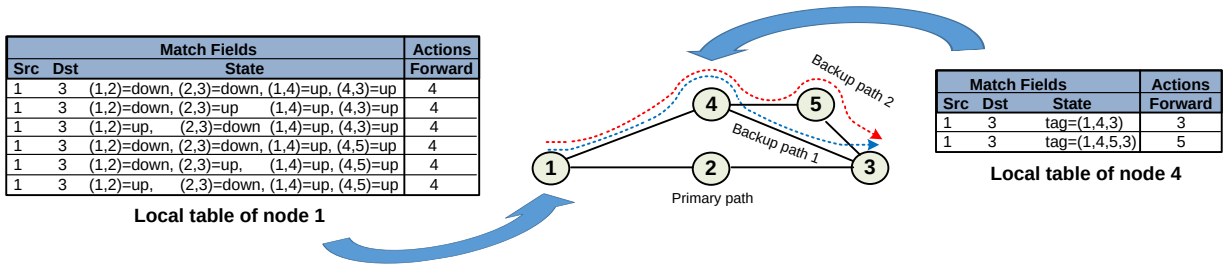


Fig. 5: The backup forwarding rules stored in the local tables of the nodes (depicted for nodes 1 and 4) before compression. The packets are matched against both their headers (source and destination node IDs) and the current state of links and tags.

node 4 for each of the two possible backup paths. This way, node 4 can decide whether to forward the packets to node 3 or 5. 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.

We note that the number of backup forwarding rules stored by the above method increases with the number of supported backup paths and the number of failed links on the primary path. On the positive side, several systematic methodologies for the design of wildcard rules have been developed [17], [18]. We can directly apply these methods to compress the local forwarding tables managed by the agents.

B. Consistent Packet Forwarding

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 Figure 5 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 third 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.

V. Evaluation Results

In this section, we conduct a preliminary evaluation of the proposed message dissemination algorithm using the topology of a real wireless mesh network depicted in Figure 6(a). This can represent a relatively static tactical ad hoc network where links fail because of depleted energy or physical layer disturbances. We plan to explore the impact of mobility in future work.

We randomly pick five critical pairs of nodes in the network ($C = 5$) and set the primary paths as the shortest paths between these nodes with respect to the hop count length (p_c sets). The traffic between a pair of nodes follows the gravity model [20]. We create two backup paths for each pair of nodes ($\mathcal{B}_c$ sets). These are the two shortest paths computed when all the links of the primary path have been removed.

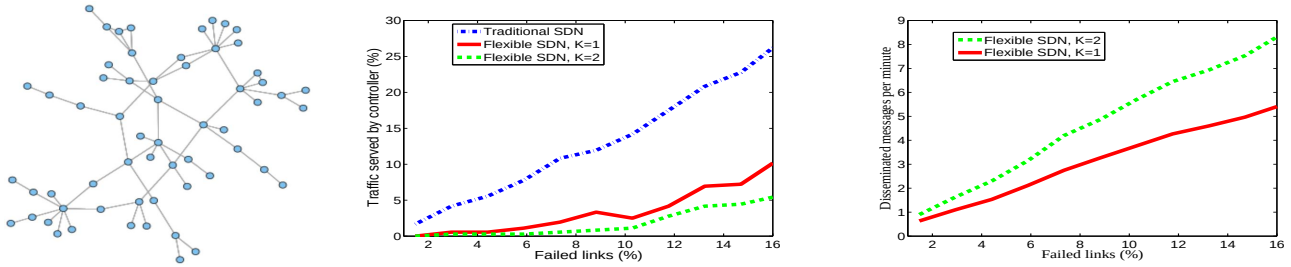
We simulate a dynamic scenario where a given percentage of randomly picked links fail each minute, and take measurements for one hour. We vary the percentage of failed links from 1% to 16% and compare the performance of the proposed system (Flexible SDN) with the traditional SDN system in which the controller manages all the traffic failures. We depict the results in Figure 6(b) when $K = 1$ or $K = 2$ backup paths are computed by the proposed algorithm for each critical pair of nodes. As expected, the traffic that reaches the controller increases with the percentage of failed links in both systems (up to 26% in the traditional system). The flexible SDN system performs better than its counterpart, especially for $K = 2$, since it directly handles a portion of the traffic using the local agents.

Next, we explore the overheads of the messages disseminated between the agents (Figure 6(c)). While the overheads increase with the percentage of failed links, the dissemination rate is no more than a few messages per minute. Supporting an extra backup path (moving from K=1 to K=2), increases the overheads by half. However, it also improves performance as we showed in Figure 6(b). This highlights the tradeoff between performance and overheads.

VI. Related Work

SDN architectures which combine centralized and distributed network control have been proposed in the past, e.g., see [21], [22]. The above works, however, target the scalability issue in large-scale data center networks that are relatively static. In contrast, we propose flexible SDN protocols that mimic basic of operations of MANETS to handle network changes in the ad hoc environment.

The idea of using stateful SDN to locally reroute data packets has been also proposed in wired networks, e.g. see [23]. While the above work also suggested the pre-planning of backup forwarding rules at the data plane nodes, it did not



(a) Wireless mesh topology. (b) Performance benefits of flexible SDN. (c) Overheads of flexible SDN.

Fig. 6: (a) The real mesh network topology in [19]. (b) The traffic that reaches the SDN controller as a function of the percentage of failed links. (c) The message dissemination overheads of Flexible SDN.

provide message dissemination and coordination mechanisms between the nodes. Hence, it can handle only one link failure.

Another method that can be used to improve the performance and reliability of SDN architecture is the deployment of many (physically distributed but logically centralized) controllers across the network [24]. As we showed in Figure 4, however, the wireless links between the controller and nodes can cause many times slower reaction to failures than our approach. Hence, the two approaches, controller placement and flexible control, should be seen as *complementary*, not as competitive.

VII. Conclusion

In this paper, we made a step towards bringing SDN into tactical ad hoc networks. Namely, we designed and implemented 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 wireless devices that are commonly used today, making our design directly applicable to the wireless market. Going a step further, we studied theoretical, yet practical, methods of managing the overheads of flexible control which are crucial for the success of these systems. In future, we plan to extend our approach beyond routing applications, to flexibly control storage and computing resources of the ad hoc network.

Acknowledgement

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. 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.

References

[1] W. Xia, Y. Wen, C. Foh, D. Niyato, "A Survey on Software-defined Networking", *IEEE Communication Surveys & Tutorials*, vol. 17, no. 1, pp. 27-51, 2015.

[2] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Holzle, S. Stuart, A. Vahdat, "B4: Experience with a Globally-Deployed Software Defined WAN", *ACM SIGCOMM*, 2013.

[3] K. Poularakis, G. Iosifidis, G. Smaragdakis, L. Tassiulas, "One Step at a Time: Optimizing SDN Upgrades in ISP Networks", *Infocom*, 2017.

[4] X. Jin, L. Erran Li, L. Vanbever, and J. Rexford, "SoftCell: Taking Control of Cellular Core Networks", *ACM CoNEXT*, 2013.

[5] X. Foukas, D. Nikaein, M.M. Kassem, M.K. Marina, K. Kontovasilis, "FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks", *ACM CoNEXT*, 2016.

[6] A. Dusia, V.K. Mishra, A.S. Sethi, "An Architecture for SD-MANET: Software Defined Mobile Ad hoc Network", *to appear in Milcom*, 2017.

[7] K. Poularakis, Q. Qin, E. Nahum, M. Rio, L. Tassiulas, "Bringing SDN to the Mobile Edge", *DAIS Workshop*, 2017.

[8] K. Poularakis, G. Iosifidis, L. Tassiulas, "SDN-enabled Tactical Ad Hoc Networks: Extending Programmable Control to the Edge", *to appear in IEEE Communications Magazine*, 2017.

[9] H. Xu, Z. Yu, C. Qian, X. Li, Z. Liu, "Minimizing Flow Statistics Collection Cost of SDN Using Wildcard Requests", *IEEE Infocom*, 2017.

[10] J. Loo, J. Lloret, J. H. Ortiz, "Mobile Ad Hoc Networks: Current Status and Future Trends", *Boca Raton, FL, USA: CRC*, 2011.

[11] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "OpenState: Programming Platform-independent Stateful OpenFlow Applications Inside the Switch", *ACM Sigcomm Computer Communication Review*, vol. 44, no. 2, pp. 44-51, 2014.

[12] M. Moshref, A. Bhargava, A. Gupta, M. Yu, R. Govindan, "Flow-level State Transition as a New Switch Primitive for SDN", *HotSDN*, 2014.

[13] Open vSwitch, http://openvswitch.org

[14] C. Koufogiannakis, N.E. Young, "Greedy Δ-Approximation Algorithm for Covering with Arbitrary Constraints and Submodular Cost", *Algorithmica, vol. 66, no. 1, pp. 113-152*, 2013.

[15] J. Kempf, S. Whyte, J. Ellithorpe, P. Kazemian, M. Haitjema, N. Beheshti, S. Stuart, H. Green. "OpenFlow MPLS and the open source label switched router", International Teletraffic Congress, 2011.

[16] S.K. Kaur, J. Singh, N.S. Ghumman, "Network Programmability Using POX Controller", *ICCS*, 2014.

[17] X.N. Nguyen, D. Saucez, C. Barakat, T. Turletti, "Rules Placement Problem in Openflow Networks: A Survey", *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1273-1286, 2016.

[18] M. Rifai, N. Huin, C. Caillouet, F. Giroire, D.M.L. Pacheco, J. Moulierac, G. Urvoy-Keller, "Too many SDN rules? Compress them with MINNIE", *IEEE Globecom*, pp. 1-6, 2015.

[19] A. Neumann, E. López, L. Navarro, "An evaluation of BMX6 for Community Wireless Networks", *CNBuB*, 2012.

[20] P. Tune, M. Roughan, "Internet Traffic Matrices: A Primer", *Recent Advances in Networking, vol. 1*, 2013.

[21] M. Yu, J. Rexford, M. J. Freedman, J. Wang, "Scalable Flow-based Networking with DIFANE", *ACM Sigcomm*, 2010.

[22] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, "Devoflow: Scaling Flow Management for High-performance Networks", *ACM Sigcomm*, 2011.

[23] A. Capone, C. Cascone, A. Nguyen, B. Sansò, "Detour Planning for Fast and Reliable Failure Recovery in SDN with OpenState", *DRCN*, 2015.

[24] B. Heller, R. Sherwood, N. McKeown, "The Controller Placement Problem", *ACM HotSDN*, 2012.