Control Plane Performance in Tactical Software Defined Networks

Joakim Flathagen and Ole Ingar Bentstuen
Norwegian Defence Research Establishment (FFI), Norway
Email: {joakim.flathagen,ole-ingar.bentstuen}@ffi.no

***Abstract*—Software Defined Networking can be beneficial in tactical networks to increase flexibility, provide programmability and to simplify management. The immense dynamics in tactical communication networks can, however, lead to excessive control traffic. As capacity in such networks is limited, this is a challenge that must be resolved. In this paper we investigate the control traffic overhead and present an analytical model that can predict the number of control messages for SDN networks with a given size and packet loss probability. We verify the soundness of the model using network emulations, and show that there is a good match between the analytical estimates of control traffic and real measurements. The tools provide some important insights that can be used to design and improve tactical SDN networks.**

I. Introduction

Software Defined Networking (SDN) is a game changing approach to build communication networks that is currently transforming both networks within data centers and service provider networks. The tactical environment can also incorporate several of the benefits from softwarization of networks such as flexibility, programmability, and management [1]. SDN networks are therefore considered beneficial both to provide Quality of Service (QoS) at the tactical edge [2] and to facilitate coalition networks [3]. Tactical networks differ, however, from traditional networks in a number of aspects such as their error prone nature and limited capacity. In this paper, we focus on investigating whether SDN can be applied to tactical networks given these characteristics.

SDN networks are built on the idea of separating the data plane and the control plane, as shown in Fig.1. The most prominent design goal of OpenFlow, which is the de-facto southbound protocol for SDN, is to keep the data plane simple and to delegate all control tasks to a centralized SDN controller [4]. As a result of this, network devices in the data plane have to consult the controller on how to handle packets and flows, which can lead to excessive traffic in the control plane. The limited capacity in a tactical network further complicates the challenge in implementing a centralized architecture.

In this paper, we tackle this challenge in the following way: First, we investigate the control overhead of OpenFlow via experiments with the ONOS (Open Networking Operating System) SDN controller [5]. Then, we develop an analytical model that can predetermine the control traffic given a particular network layout and packet loss probability. Finally, we verify the analytical model by conducting experiments with ONOS and the network emulator Mininet with OpenVSwitch software switches. We also present how the model can be used to investigate the effectiveness of an alternative topology discovery method. Before presenting our model and results, we review the related works.

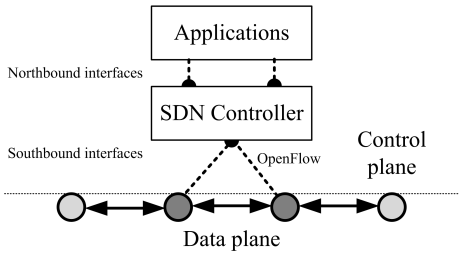


Fig. 1. SDN architecture.

II. Related works

Bianco et al. investigated the amount of control traffic generated by the default forwarding mechanisms in ONOS [6]. They found that the control messages in a reactive flow paradigm can be considerable. One method that contributes to reduce the control traffic in such networks is Control Message Quenching [7]. Here the control traffic is reduced for unknown flows by letting the switch maintain a table with all unique source-destination pairs.

In the work [8] the authors revealed a number of scalability problems of OpenFlow. Consequently, they proposed Devoflow to process flow rules more efficiently by managing short-lived paths in the datapath, while long-lived paths are handled by the controller. Despite reducing control plane traffic, both [7] and [8] require modifications of OpenFlow.

The seminal work B4 [9] avoided some of the OpenFlow overhead by using proactively established flows instead of reactive flows. They also reduced the flow statistics frequency and only requested statistics for a smaller number of flows.

One method to improve the scalability and reliability is to employ multiple controllers. Muqaddas et al. developed some empirical models to quantify the traffic exchanged among multiple SDN controllers in ONOS [10]. We don't consider multiple controllers in our paper. However, both [10] and [6] share resemblance to our work considering that we focus on ONOS.

Pakzad et al. proposed a new topology discovery method that contributes to reduce some of the control traffic [11]. We analyze the efficiency of this particular method in our paper. Spencer et al. evaluate the control plane for tactical networks [12]. They mainly discuss reactive networks and we

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/MILCOM.2018.8599722

thus consider it complementary to our work. However, none of the above works consider an error-prone data plane subjected to packet loss, whereas this is the goal of our paper.

III. Control traffic in a steady-state SDN

In this section we evaluate the control traffic in an OpenFlow SDN controlled by the ONOS SDN controller. The analysis is based on measuring the control traffic in a set of basic networks. The networks are based on OpenVSwitch switches within the Mininet emulator, and the traffic is measured using a network sniffer between the ONOS controller and the Mininet nodes. In addition, we have analyzed the ONOS source code and consulted the OpenFlow documentation. Before presenting our model, it is worth defining the parameters characterizing a generalized SDN.

A. Network definition

The data plane of an SDN network can be modeled as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of links. Given is a set of switches $\mathcal{S} \subseteq \mathcal{V}$ and a set of clients (or users) $\mathcal{U} \subseteq \mathcal{V}$, where $\mathcal{S} \cap \mathcal{U} = \emptyset$. We denote the number of switches $\mathcal{N} = |\mathcal{S}|$, and the number of clients $n = |\mathcal{U}|$. Each switch $s \in \mathcal{S}$ has a number of ports that can be connected either to a client or a switch in the data plane. We define a client-connected port as an external port and a switch-connected port as an internal port. Let i_s be the number of internal ports for switch s and e_s the corresponding number of external ports. A link between switches s_1 and s_2 is denoted v_1, v_2, whereas a link between s_1 and client u_1 is denoted v_1, u_1. Finally, we consider a control plane that is logically and physically separated from the data plane. The control plane consists of one single controller $\mathcal{C}$ that is directly connected to all switches in $\mathcal{S}$.

B. Model for a steady-state network

Based on the above parameters, we can model the control plane traffic. We define steady-state as a situation without network dynamics. Hence, the SDN controller utilizes OpenFlow merely to collect network information and not to install or alter flows. For the sake of clarity, we consider the steady-state first, and discuss the dynamic case later in this paper.

The task at hand is to estimate the total number of exchanged OpenFlow (OF) messages for a given topology $\mathcal{G}$ with $\mathcal{N}$ switches. Table I lists the OF messages that are present in the control plane and the typical timer intervals used in ONOS. The default intervals used in ONOS are denoted "Short", while we also explore the effect of prolonged intervals (denoted "Long") later in the paper.

Considering a steady-state network, the expected OF packet rate (i.e., the control plane load) is the sum of the individual OF message packet rates. A prediction of this rate is therefore relatively straightforward once the contribution of each individual rate is understood. Most of the OF message exchanges between the controller and a switch are of the request/response type, meaning that two OF messages (one request and one response) are exchanged over the control channel for each timer interval. Both Port, Flow, Table and Meter status messages are exchanged in this manner, i.e., given $\mathcal{N}$ switches, the rates for these are (in packets per second):

TABLE I
OF messages between a controller and a switch.

OpenFlow message	Intensity	Interval	Short	Long
OFPMP_PORT_STATUS	λ_p	t_p	5 s	60 s
OFPMP_FLOW_DESC	λ_f	t_f	5 s	60 s
OFPMP_TABLE_STATUS	λ_t	t_t	5 s	60 s
OFPMP_METER	λ_m	t_m	10 s	60 s
OFPMP_GROUP	λ_g	t_g	10 s	60 s
Link Layer Discovery Protocol	λ_l	t_l	3 s	60 s
OFPT_ECHO_REQUEST	λ_e	t_e	5 s	5 s

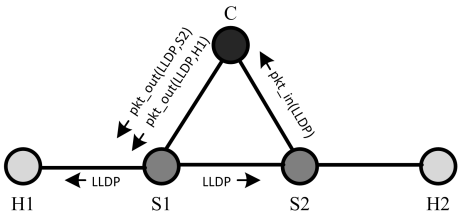


Fig. 2. LLDP protocol.

$$\lambda_p = 2 \cdot \mathcal{N} \cdot t_p^{-1} \text{ pps} \quad (1)$$

$$\lambda_f = 2 \cdot \mathcal{N} \cdot t_f^{-1} \text{ pps} \quad (2)$$

$$\lambda_t = 2 \cdot \mathcal{N} \cdot t_t^{-1} \text{ pps} \quad (3)$$

$$\lambda_m = 2 \cdot \mathcal{N} \cdot t_m^{-1} \text{ pps} \quad (4)$$

Notice that in ONOS, OFPMP_FLOW_DESC and OFPMP_TABLE_STATUS requests are controlled by the same timer, i.e., $\lambda_f = \lambda_t$. For each t_g interval, four messages are exchanged such that:

$$\lambda_g = 4 \cdot \mathcal{N} \cdot t_g^{-1} \text{ pps} \quad (5)$$

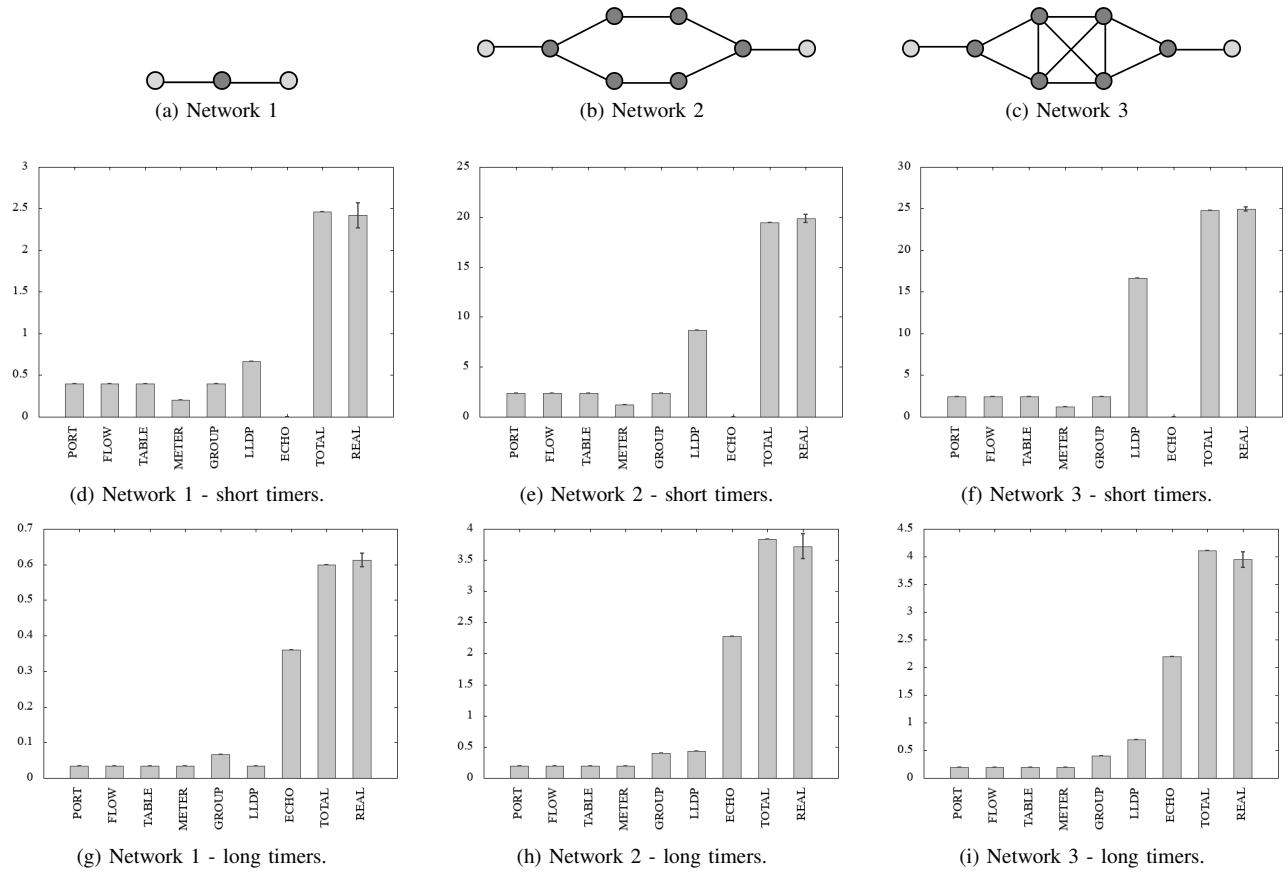
The Link Layer Discovery Protocol (LLDP) is not of the request-response type as the messages discussed above. In addition to the interval t_l, the LLDP rate depends on the number of active ports on the switches. For each internal port of a switch (i_s) one *pkt_in* and one *pkt_out* are transmitted over the control channel. For each external port (i.e., port connected to a host), only one *pkt_out* message is transmitted (since hosts do not reply to LLDP messages), see Fig. 2. The rate can be expressed as:

$$\lambda_l = \sum_{s \in \mathcal{S}} t_l^{-1} \cdot (2 \cdot i_s + e_s) \text{ pps} \quad (6)$$

Echo request messages are initiated by the switches to verify proper connectivity with the controller. According to the OpenFlow documentation, the controller can also initiate these messages, but we have not seen this in ONOS and it is not considered here. For OpenVSwitch switches, the echo request interval is typically 5 s, but echo-requests are not transmitted if other OF messages verify the connection within the interval t_e. The rate λ_e depends, in other words, on the combined intensity of the other OF message rates. The expected OF message intensity from the controller λ_c is:

$$\lambda_c = t_l^{-1} + t_p^{-1} + t_f^{-1} + t_g^{-1} + t_m^{-1} \text{ pps} \quad (7)$$

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/MILCOM.2018.8599722



(a) Network 1 (b) Network 2 (c) Network 3

(d) Network 1 - short timers. (e) Network 2 - short timers. (f) Network 3 - short timers.

(g) Network 1 - long timers. (h) Network 2 - long timers. (i) Network 3 - long timers.

Fig. 3. Control traffic (in packets per second) for three different network configurations

Notice that t_t is omitted as λ_t is controlled by t_f in ONOS. Now, λ_c can be modeled as a Poisson process. The probability for a switch transmitting an echo request is equal to the probability of receiving no OF messages within the interval t_e, which is $\mathcal{P}(T > t_e) = e^{-\lambda_c t}$. The intensity of echo messages in the control plane is hence:

$$\lambda_e = \frac{2 \cdot \mathcal{N} \cdot e^{-\lambda_c \cdot t_e}}{t_e} \text{ pps} \tag{8}$$

Finally, the steady-state OF rate is the sum of the individual message rate components defined above:

$$\lambda_{ss} = \lambda_p + \lambda_f + \lambda_r + \lambda_m + \lambda_g + \lambda_l + \lambda_e \text{ pps} \tag{9}$$

C. Verification of the model

The model presented in Eq. 9 can be used to investigate the specific contribution each OF message has on the total control overhead for a particular network configuration. To verify the model, let's consider three different networks with varying number of nodes and interconnections. The three configurations are shown in the Figures 3a, 3b and 3c. Here $\mathcal{N}$ is 1, 6 and 6 for the three different configurations, and the corresponding numbers of links $|\mathcal{E}|$ are 2, 8 and 12.

The estimates for the individual packet rate intensities are calculated for each of the three networks. For each of the networks, we consider both the default ONOS timers (short), which are presented in the Figures 3d, 3e and 3f, and an alternative setup using long timers, presented in the Figures 3g, 3h and 3i. We refer to Table I for the values employed.

For each of the six combinations presented in Fig. 3, the estimates are verified by corresponding network emulations. We used ONOS 1.10 and Mininet with OpenVSwitch v 2.8.0 switches. The OpenFlow version is 1.3. The OpenFlow packets are measured on the control plane interface between ONOS and Mininet using Wireshark. For each setup, we ran 10 individual experiments. The average OF intensity λ_{ss} (in packets per second) and the standard deviation is given in the figures (denoted as "REAL"), is comparable with the estimated "TOTAL". The results show that the model can exactly predict the rate of OF messages in steady-state.

D. Application of the model

The model can be applied to carefully adjust network timers to a particular network configuration. In our examples, it is interesting to observe that for short timers, the echo message rate is negligible as the overall OF rate leads to $\lambda_e \approx 0$, whereas for long timers, the echo rate contributes to a substantial amount of the total OF rate. This leads us to the intermediate conclusion that for a narrowband tactical network, merely adjusting OF timers is not sufficient. It might be necessary to alter, for example the echo protocol, in order to utilize the control channel effectively.

One possible application of the model could be to investigate the effect of various control message reduction schemes found in the literature. One such example is the mechanism proposed in [11], which optimizes the LLDP protocol. Instead of creating a unique LLDP packet for each port of each switch,

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/MILCOM.2018.8599722

and sending each such packet to the corresponding switch via a separate OpenFlow Packet-In message as shown in Fig. 2, the method sends only a single LLDP packet to each switch. Using this method, e_s is now irrelevant and Eq. 6 can therefore be reduced to $\lambda_l = \sum_{s \in \mathcal{S}} t_l^{-1}(1 + i_s)$ pps.

By applying this modification to our model and recalculating the estimates for the six networks shown in Fig. 3, we can evaluate the effectiveness of the proposal. In Fig. 4 we see that the modified LLDP is most efficient in network with many interconnections and short timers.

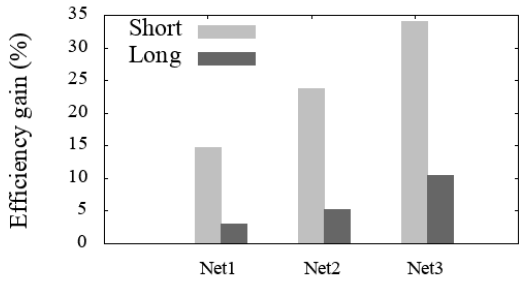


Fig. 4. Efficiency gain in applying the Topology discovery method in [11] for three different network configurations with short and long OF timers.

Other proposals involving packet quenching [7] or other bandwidth saving techniques can be applied to the model in a similar manner.

IV. Control traffic during flow generation

The flow generation method is ignored in the steady-state OF rate presented above. Although this is a simplification, the model is applicable as long as the network is stable. However, a tactical network is dynamic, and we therefore need to consider the overhead of the flow generation method. In particular we need to determine the number of OF messages exchanged for each new flow installed in the network. In the following, we discuss two alternative methods that functionality-wise implements a large distributed switch; first a reactive approach as used by ONOS Reactive Forwarding (*fwd*), and a proactive approach, as used in the ONOS Intent framework.

A. Reactive flow generation

Flow generation with the *fwd* application is previously studied extensively by Bianco et al [6], and we base the reasoning in this section on their paper. Lets consider a simple network similar to the one in Fig. 2 consisting of two hosts H_1 and H_2 and two switches S_1 and S_2. Upon communication between the two hosts, the flow is reactively established using ARP as outlined in Table II. Notice that some of these messages could be omitted (O), if proxy ARP is employed. But to simplify, we assume that proxy ARP is disabled.

The process shown in Table II consists of three stages. First the discovery stage, then a stage where some IP-packets are forwarded in the control plane, before the third stage where the flows are installed. The number of OF messages exchanged for setting up a bidirectional path between two hosts u_1, u_2 is given by [6] as:

$$N_{OFr} \in (17 \cdot d(u_1, u_2), 2 \cdot \mathcal{L} + \mathcal{N} + 15 \cdot d(u_1, u_2)) \quad (10)$$

TABLE II
Packet sequence during flow setup. R denotes reactive flow setup, where O means that the message can be omitted with Proxy ARP. P denotes proactive setup.

Stage	Comment	Direction	Packet
1	R	$H_1 \rightarrow S_1$	ARP-REQ:H_2?
1	R	$S_1 \rightarrow C$	pkt_in (ARP-REQ:H_2?)
1	R,O	$C \rightarrow S_1$	pkt_out (ARP-REQ:H_2?):flood
1	R,O	$S_1 \rightarrow S_2$	ARP-REQ:H_2?
1	R,O	$S_2 \rightarrow C$	pkt_in (ARP-REQ:H_2?)
1	R,O	$C \rightarrow S_2$	pkt_out (ARP-REQ:H_2?):flood
1	R,O	$S_2 \rightarrow H_2$	ARP-REQ:H_2?
1	R,O	$H_2 \rightarrow S_2$	ARP-REP:H_2
1	R,O	$S_2 \rightarrow C$	pkt_in (ARP-REP:H_2)
1	R	$C \rightarrow S_1$	pkt_out (ARP-REP:H_2):port 1
1	R	$S_1 \rightarrow H_1$	ARP-REP:H_2?
2	R	...	Initial IP packets $H_1 \rightarrow H_2$ via C
3	R,P	$C \rightarrow S1$	flow_mod($H_1 \rightarrow H_2$)
3	R,P	$C \rightarrow S1$	barrier_request
3	R,P	$S1 \rightarrow C$	barrier_reply
3	R,P	$C \rightarrow S2$	flow_mod($H_1 \rightarrow H_2$)
3	R,P	$C \rightarrow S2$	barrier_request
3	R,P	$S2 \rightarrow C$	barrier_reply
3	R,P	$C \rightarrow S1$	flow_mod($H_2 \rightarrow H_1$)
3	R,P	$C \rightarrow S1$	barrier_request
3	R,P	$S1 \rightarrow C$	barrier_reply
3	R,P	$C \rightarrow S2$	flow_mod($H_2 \rightarrow H_1$)
3	R,P	$C \rightarrow S2$	barrier_request
3	R,P	$S2 \rightarrow C$	barrier_reply

The above shows the lower and upper bound for the number of OF messages. We have defined (i_s) as the number of internal ports for switch s. $\mathcal{L}$ refers to the number of links connecting the switches, which is $\mathcal{L} = \sum_{s \in \mathcal{S}} i_s/2$. For our evaluation, it is more practical to use the average shortest path in the network, $\overline{\mathcal{D}_{sp}}$. This is defined as $\overline{\mathcal{D}_{sp}} = 1/(n \cdot (n-1)) \cdot \sum_{i \neq j} d(u_i, u_j)$, where $d(u_i, u_j)$ denote the shortest distance between i and j. We define the number of switches in $\overline{\mathcal{D}_{sp}}$ as $\overline{S_{sp}}$ and the number of OF managed links as $\overline{L_{sp}}$.

B. Intent based proactive flow generation

An alternative to the reactive scheme is to use the intent framework. It provides a way to generate flows in a proactive manner. Let us again relate to a network of two hosts and two switches and the flow setup sequence in Table II. For such a proactive scheme, only stage 3 is relevant since no ARP messages are flooded. Instead of ARP discovery, an intent (i.e., an end to end flow) must be proactively determined by a network administrator. Methods to establish intents reactively exist [6], but are not considered here.

From Table II we can derive that the number of OF messages exchanged for setting up a bidirectional path between two hosts u_1, u_2 is:

$$N_{OFi} = 6 \cdot (d(u_1, u_2) - 1) \approx 6 \cdot \overline{S_{sp}} \quad (11)$$

Notice that the third sequence shown in Table II can be optimized by the controller by concatenating flow_mod messages and barrier requests. The minimum number of OF messages per flow setup (or pairwise connection) is in this case reduced to four OF messages per switch. This is also the case for stage 3 in reactive forwarding. Since this behavior

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/MILCOM.2018.8599722

is not deterministic, we use 6 OF messages in the following discussion.

An initial comparison of the two flow setup paradigms can be performed. Since N_{OFr} at the lower bound is $17\,\overline{\mathcal{D}_{sp}}$ and N_{OFi} is $6\,(\overline{\mathcal{D}_{sp}}-1)$ we can immediately derive that the latter is more effective than the former for flow setup. Notice also that a reactive paradigm immediately removes flow rules for inactive flows. Considering that the average length of a flow in the Internet is very short, around 20 packets per flow [8], the number of control packets can very likely exceed the number of data packets per flow. For an intent based proactive paradigm, this issue is mitigated, since flows rules are active until a network manager deletes them. The number of pre-installed flows can, however, reach $\mathcal{F}_{max} = n(n-1)$, which equals the maximum number of pairwise connections in $\mathcal{G}$. This means that the number of active flows to maintain can be overwhelming. A few available methods can be used to reduce the number of flows in a proactive network, for example by grouping clients using Virtual Private LAN Service (VPLS) and making associated flow rules for the VPLSs. This can make the intent-based scheme very attractive for a tactical SDN. Based on the above reasoning, we only investigate the intent-based case for the remaining of this paper.

V. Flow generation in an error-prone network

We have now determined the steady-state load and the flow setup cost. In this section we study the above features in a network with packet loss. Although the packet loss can be caused by weak radio signals, on-air collisions, full buffers etc, we do not distinguish on cause in this section.

A. Prediction model

The LLDP protocol is used in SDNs to discover network neighbors and the potential loss of such a neighbor using the notion of links. A link $i,j \in \mathcal{E}$ and is said to be available if i and j are directly connected and can communicate with each other. If no LLDP messages are received over the link i,j within the time interval t, the link is defined as "down". Considering a packet loss probability of p_{ij} and an LLDP probe interval of t_l, the probability for determining a link (ij) as "down" is:

$$p_{ij}^{\left\lfloor \frac{t}{t_l} \right\rfloor} \tag{12}$$

Assume that packet loss probabilities for all $(i,j) \in \mathcal{E}$ is p. For an intent-based scheme, the probability that an end-to-end flow between clients u_1 and u_2 is considered "down" within the time interval t is equal to the probability that a fault is discovered in any of the links constituting the flow path. This is equal to one minus the probability that no faults is discovered within the time interval:

$$\mathcal{P}(x>0) = 1 - \mathcal{P}(0) = 1 - (1 - p^{\left\lfloor \frac{t}{t_l} \right\rfloor})^{\overline{L_{sp}}} \tag{13}$$

If one of the links in a path is broken, the controller removes all flows in the path before reinstalling them using an available and alternative path. We have already estimated the number of OF messages required to install bidirectional flows on one switch. The flow removal process is also similar. Given that all end-to-end flows (or intents) are represented in $\mathcal{F}$, the rate of control messages caused by link losses (λ_p) can be estimated:

$$\lambda_p = \frac{1}{t} \sum_{f_{uu} \in \mathcal{F}} 1 - (1 - p^{\left\lfloor \frac{t}{t_l} \right\rfloor})^{\overline{L_{sp}}} \cdot 2 \cdot N_{OFi} \text{ pps} \tag{14}$$

In a proactive network, we assume that there exist intents between all node pairs, i.e., that F includes all possible client-to-client combinations. By substituting N_{OFi} with Eq.11 we get:

$$\lambda_p = \frac{6n(n-1)}{t} \cdot \overline{S_{sp}} \cdot 1 - (1 - p^{\left\lfloor \frac{t}{t_l} \right\rfloor})^{\overline{L_{sp}}} \text{ pps} \tag{15}$$

The expected OF intensity $\lambda = \lambda_{ss} + \lambda_p$. Notice that λ_l in Eq. 6 is also affected by packet loss in the data plane such that:

$$\lambda_l = \sum_{s \in \mathcal{S}} t_l^{-1} \cdot (2 \cdot (1-p) i_s + e_s) \text{ pps} \tag{16}$$

B. Evaluation

We evaluate the above prediction model using a network of ten clients and six switches as shown in Fig. 5. The number of interconnections (intents) are here 50. We consider the default (short) ONOS timers for OpenFlow as presented in Table I. One method to improve the network's ability to adapt to dynamics (for example persistent packet loss) is to reduce the LLDP interval t_l. However, this will also lead to increased OF intensity. An alternative method is to reduce t, i.e., the tolerance to lose LLDP packets. We investigate the default setting of $t = 10$ s and an alternative setting of $t = 7$ s in the following experiments.

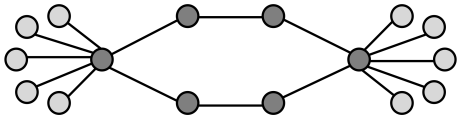
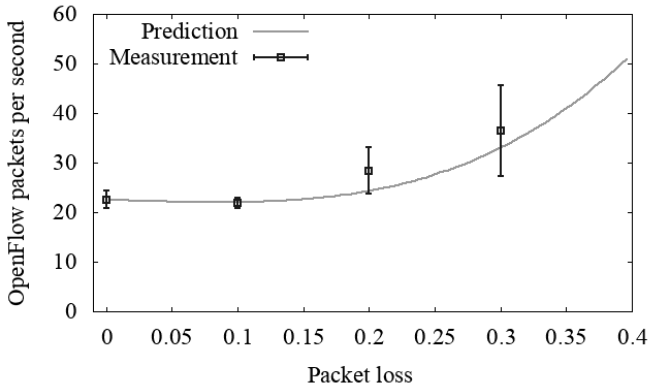


Fig. 5. Network configuration using ten clients and six switches.

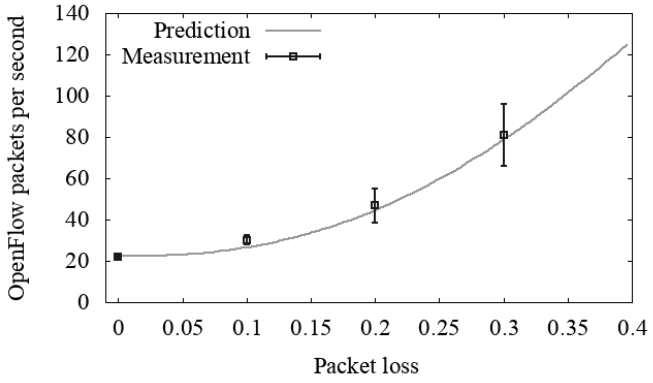
In Fig. 6a the predicted OF intensity λ is presented varying on packet loss probability p as in Eq. 15 while using $t = 10$. In Fig. 6b, the prediction is plotted for $t = 7$. The predictions are verified by network emulations with ONOS and Mininet, conducted as previously described. Five separate emulations of 300 s each are performed for $p = \{0.1, 0.2, 0.3\}$. The standard deviation is presented in the figures.

We see a good match between the measurements and the prediction model. The downward trend between $(0, 0.1)$ in Fig. 6a is due to loss of LLDP packets in the data plane (Eq. 16) which again reduces the number of *pkt-in* LLDP packets in the control plane. At the same time the loss is not high enough to foster flow reinstallment as with an increased p. The model has a slight tendency to underestimate the packet rate, but overall, the measurements show that the prediction model show sufficient accuracy for its intended purpose.

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/MILCOM.2018.8599722



(a) $t = 10$



(b) $t = 7$

Fig. 6. Comparison of analytic model prediction and real measurements for intent-based flow generation with packet loss in the data plane.

VI. Discussion

In this paper, we have investigated the rate of OpenFlow messages in the control plane. Based on our model and experiments, we can derive some conclusions and discuss a few directions for future research that can be helpful in designing tactical SDNs.

First, the various OpenFlow timers used by the SDN controller must be fine-tuned and carefully adjusted to balance OF overhead and the ability to tackle network dynamics.

Second, methods to replace the typical request response nature of OpenFlow should be investigated. The OF rate in steady state λ_{ss} can be considered pure overhead, and a publish/subscribe scheme that informs the controller only upon changes in the data plane can reduce this overhead considerably. One way could be to exchange switch state information piggybacked on Hello-messages.

Third, the ability for a switch to perform local repair of some flow-rules can be beneficial. The intent-based framework studied in this paper requires that the complete end-to-end intent is recompiled upon link break. A mechanism that delegates some of the flow computation to the switches could be beneficial in a tactical network, since this will increase agility and reduce the flow setup overhead. Fourth, a reduction in the number of installed flow rules can be conducted by grouping clients in VPLS. This makes the network more scalable, and it is simpler for an administrator to proactively determine clients-to-VPLS memberships and associated flow rules than to implement client-to-client flow rules for all possible client combinations.

The effect of implementing the above improvements or modifications can be predicted for a given network topology with the tools presented in this paper.

VII. Conclusion

This paper has provided tools to investigate the control plane load in tactical OpenFlow based SDNs. The soundness of these tools are verified with network emulations proving a good match between the analytical estimates of OpenFlow traffic and real measurements. The knowledge provided by these tools provides some important insights that can be used to design and improve tactical SDNs. Although we use ONOS in our work, the methods can be applied to other SDN controllers.

The methods consider a separate control plane and data plane. This assumption can't be guaranteed in a tactical network. Although the conclusions derived from the methods can still be useful in such networks, future work should include extending the model to an unreliable and in-band control channel.

References

[1] J. Nobre, D. Rosario, C. Both, E. Cerqueira, and M. Gerla, "Toward software-defined battlefield networking," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 152–157, 2016.

[2] K. Phemius, J. Seddar, M. Bouet, H. Khalifé, and V. Conan, "Bringing SDN to the edge of tactical networks," in *MILCOM*. IEEE, 2016, pp. 1047–1052.

[3] V. Mishra, D. Verma, and C. Williams, "Leveraging SDN for Cyber Situational Awareness in Coalition Tactical Networks," in *IST-148*, 2016.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM*, vol. 38, no. 2, pp. 69–74, 2008.

[5] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "ONOS: towards an open, distributed SDN OS," in *HotSDN*. ACM, 2014, pp. 1–6.

[6] A. Bianco, P. Giaccone, R. Mashayekhi, M. Ullio, and V. Vercellone, "Scalability of ONOS reactive forwarding applications in ISP networks," *Computer Communications*, vol. 102, pp. 130–138, 2017.

[7] T. Luo, H.-P. Tan, P. C. Quan, Y. W. Law, and J. Jin, "Enhancing responsiveness and scalability for OpenFlow networks via control-message quenching," in *ICTC*. IEEE, 2012, pp. 348–353.

[8] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: scaling flow management for high-performance networks," in *SIGCOMM*, vol. 41, no. 4. ACM, 2011, pp. 254–265.

[9] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined WAN," in *ACM SIGCOMM*, vol. 43, no. 4. ACM, 2013, pp. 3–14.

[10] A. S. Muqaddas, P. Giaccone, A. Bianco, and G. Maier, "Inter-controller Traffic to Support Consistency in ONOS Clusters," *IEEE Trans. on Network and Service Management*, vol. 14, no. 4, pp. 1018–1031, 2017.

[11] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, "Efficient topology discovery in Openflow-based Software Defined Networks," *Computer Communications*, vol. 77, pp. 52–61, 2016.

[12] J. Spencer, R. Taylor, and R. Hancock, "Evaluation of software-defined networking control plane performance in deployed military communications systems," in *ICMCIS*. IEEE, 2017, pp. 1–7.

Dette er en postprint-versjon / This is a postprint version.
DOI til publisert versjon / DOI to published version: 10.1109/MILCOM.2018.8599722