Towards Software Friendly Networks

Kok-Kiong Yap Te-Yuan Huang Ben Dodson Monica S. Lam Nick McKeown
Stanford University
{yapkke,huangty,bjdodson,lam,nickm}@stanford.edu

ABSTRACT

There has usually been a clean separation between networks, and the applications that use them. Applications send packets over a simple socket API; the network delivers them. However, there are many occasions when applications can benefit from more direct interaction with the network, to observe more of the current network state, and have more control over its behavior. In this paper we explore some of the potential benefits of closer interaction between applications and the network. We exploit the emergence of so-called "software-defined networks" (SDN) built above network-wide control planes, and explore how to build a more "software friendly network". We present results from a preliminary exploration that provide network services to applications via an explicit communication channel.

Categories and Subject Descriptors

C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks—*Distributed Systems; Network operating systems*

General Terms

Design, Management

Keywords

Software-Defined Networks, OpenFlow, Network OS, Network Services

This research is supported in part by the NSF POMI (Programmable Open Mobile Internet) 2020 Expedition Grant 0832820, Stanford Clean Slate Program, Google, Xilinx, Cisco, NEC, DT, DoCoMo and the Mr. and Mrs. Chun Chiu Stanford Graduate Fellowship.

APSys 2010, August 30, 2010, New Delhi, India.

1. INTRODUCTION

Part of the success of the Internet undoubtedly comes from the simple and consistent interface between applications and the network. Most applications use the socket API to request a connection to a remote computer, then simply send data into and out of the socket. The application needs no knowledge of the topology or current state of the network, and needs no control over how packets are delivered. The network is a set of dumb pipes.

However, many applications can benefit from a richer interface to the network with more visibility of its state, and more control over its behavior. In general, past efforts to increase the richness of the API have not been very successful (e.g. RSVP [12]). While many applications could benefit from the QoS control RSVP offers, few applications find RSVP optimized for their needs. For example, Skype employs a number of its own proprietary tricks to figure out the current quality and state of the network, and then uses multiple paths and rates to optimize its behavior [13]. This may be feasible for Skype, but is beyond the reach of many smaller applications.

One extreme approach to increasing the richness of the interface would be to put application-specific support directly into the network, as proposed in Active Networking [15]. Active networks exploit the fact that the network knows its own state, and attempts to expose state and control to user applications. However, the particular approach proved unpopular for several reasons, most notably because of security (preventing malicious use of the network), isolation (protecting one application's behavior from another) and performance (programmable elements slow-down the forwarding path).

Software defined networks (SDN) are emerging as a new (but backwardly compatible) way for networks to be architected. SDNs are being deployed in data centers now, and we expect them to be deployed in enterprise, campus and WAN networks in the next few years. It's therefore interesting to think about how, in light of this trend, the interface between applications and the network may change.

An SDN has the following elements (see Fig. 1): (1) A packet-forwarding datapath controlled by a narrow open vendor-independent API (e.g. OpenFlow [9]). These are the switches, routers and access points through which packets pass. (2) A network-wide operating system to control the datapath. The network OS (e.g. NOX [5]) has a global view of the network state, and has full programmatic control of the forwarding. (3) "Network features" are hosted on the network OS, to implement various network services such as routing (e.g. OSPF, BGP, multicast, multipath),

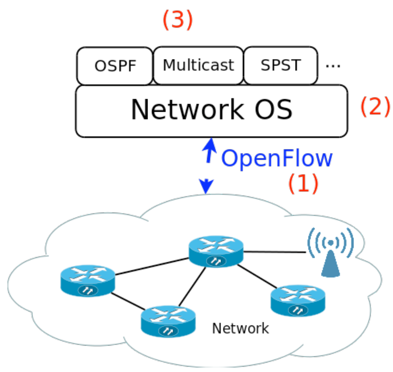


Figure 1: Components of SDN

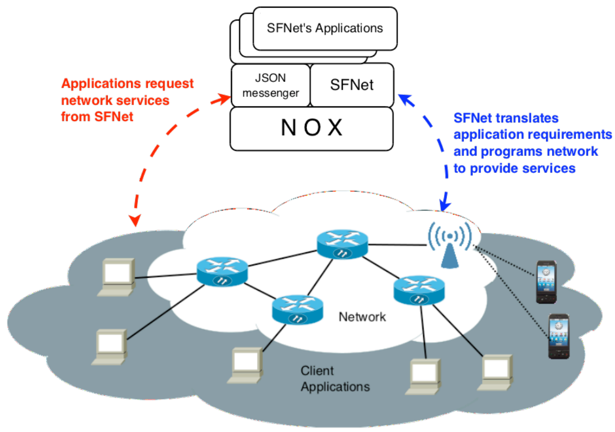


Figure 2: Architecture of SFNet

mobility management, QoS control, etc. An SDN is simply a repartitioning of the way networks are built. Initially, we expect them to support many of the features in today's networks. The key difference is that it is much easier to add *new* features to an SDN; the owners and operators of networks can improve their networks without having to wait for vendors and standards bodies. We can therefore expect SDNs to evolve and improve at a much faster pace than today's networks.

Current SDN's (and NOX in particular) don't specify how applications should interact with the network. An application may continue to use the minimal socket API, and continue to view the network merely as a means for interconnection. The question we are most interested in answering is: *How will applications interact with the network in a world where owners and operators are free to add new functionality to the control plane?* Because SDN is still in its infancy, our work is just a first step towards answering this question.

One possible outcome is that every application will provide its own "plugin" to the network OS to view and control the network, and to also define its own application-specific communication protocol to the plugin. For example, a plugin optimized for Skype might interface directly with the network OS to set up paths, reserve bandwidth, and create access control rules. Alternatively, over time, a relatively small number of "*de facto* standard" plugins might emerge for common tasks (e.g. a plugin for multicast, another for multipath routing, and yet another for bandwidth reservations). A third scenario is where plugins emerge to suit certain classes of applications (e.g. a plugin for chat applications, another for real-time video, and a third for low-latency applications). Of course, all three models can co-exist: Many applications may choose to use common feature plugins, whereas others can create their own. Our goal here is not to propose, or mandate, that any particular model will emerge. We merely make the observation that SDN allows both models, and for each application to choose its own path. The "winning features" will be picked by adoption, rather than by standards bodies.

In this short paper we explore one possible path to more "software friendly networks". We assume that applications will set up an explicit communication controller to a plugin hosted by the network control plane, allowing applications to query the network state, and issue requests.

We motivate our study using a multi-way high-definition peer-to-peer video conferencing application for mobile users. We assume that the application needs some services from the network: a known data-rate, multicast, encryption, and the means to include generic middle boxes for data transcoding. While these services may be able in some form from network boxes today, our goal here is to explore the desired API from the application's viewpoint, rather than from the network.

This enables interesting APIs, such as a congestion enquiry (which we discuss in §3). We also show that classical bandwidth reservation(§4) and request for multicast session (§5) can be provided through the same mechanism.

2. IMPLEMENTATION

To provide API to applications through a common framework, we present our prototype implemented on top of NOX, which we will refer to as SFNet (in Fig. 2). As we have previously mentioned, SFNet establish an explicit communication between applications and the network controller to expose common reusable API.

A key role of SFNet is to translate the communication between applications and the software-defined network. Meaning SFNet hides the lower network protocols from the applications, i.e., it accepts high-level requests from applications and translates into network primitives. This means application writers will not be burdened with the menial networking tasks, such as route calculation or discovering network topology. Otherwise, the barrier to entry for using these network services would be forbidding for most programmers, limiting utility and uptake of the services. In our implementation, the requests are expressed in JSON (Fig. 3), which is a simple and concise data format supported by most modern programming languages. SFNet exploits the global view provided by NOX making it easy to support high-level primitives, such as to satisfy the requests of network status and implement resource reservation.

To exploit the functionalities of SFNet, applications need to discover the location of SFNet's controller. In our implementation, our controller replies the IP address(es) of the machine it resides on when a JSON discovery request is sent by the host into the network. Having control over the entire network, we are able to make this discovery process without using broadcast.

```
{   "type" : "sfnet",
    "request" : "congestion",
    "source" : "10.79.1.110",
    "destination" : "10.79.1.111" }

{   "type" : "sfnet",
    "request" : "congestion",
    "congested" : true }
```

Figure 3: SFNet's congestion request and reply in JSON

3. CONGESTION AVOIDANCE BY DELAY-TOLERANT BACKOFF

In this section, we illustrate how SFNet provides network congestion status to applications, and show how that can be used by applications to backoff. Here, the application composes a query as a JSON message to inquire about the congestion state of the path between two hosts (Fig. 3). SFNet would then determine the switch and port on which the hosts are connected and the shortest path between them. The congestion state (i.e., percentage of link bandwidth in use) of the links, including of those between switches and hosts, are then determined. A reply is then generated for the enquiry using a pre-defined threshold.

The congestion state information is useful for a variety of purposes. For example, delay-tolerant backup can backoff when the network is congested to relieve the network for delay sensitive traffic. It can also be useful for system administrators who want to schedule backup during times for which the network is uncongested, and avoid congesting the network with backup for someone pulling an all-nighter for an impending deadline.

To validate our congestion enquiry implementation, we create a background task to generate a continuous traffic load on the network. Here, we used a TCP iperf that will try to saturate the route. We write two applications; they both transfer a 10 MB file, but the first sends it right away, and the second inquires the state of congestion from SFNet at 10 s interval, and transmits the data only if there is no congestion.

Our results shows that the file can be transferred in an average of 18.7 s with standard deviation 0.2 s without congestion (Fig. 4). With congestion and without SFNet, the transfer took an average of 38.4 s with standard deviation 1.9 s. With SFNet, the transfer is completed in an average of 18.5 s with standard deviation of 0.3 s, showing not significant difference from an uncongested network.

Our key result here is not how much faster the file transfer went but that we can effectively determine the congestion state of routes, allowing an application to backoff during congestion.

4. BANDWIDTH RESERVATION BY STREAMING SERVICE

We now describe how SFNet supports bandwidth reservation as an example. The application can send to SFNet a request for guaranteed bandwidth. SFNet determines the route and tries to map the request to appropriate bandwidth guaranteed queues in the OpenFlow network. The network provisions the required bandwidth along the route for the application at hand, to provide the best user experience possible. SFNet responds with either granted or denied to the application, depending on the availability of the requested bandwidth.

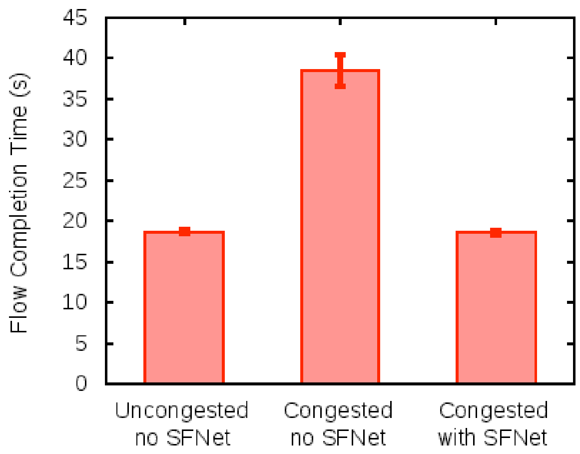


Figure 4: Flow completion time with(out) SFNet's congestion enquiry

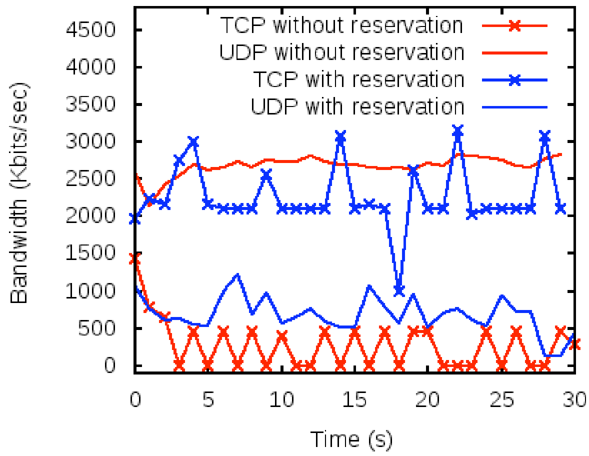


Figure 5: Available TCP bandwidth with(out) SFNet's reservation

Many interactive applications, such as video on demand and VoIP calls, are delay intolerant and has a strict bandwidth requirement. Even with traffic classification tools, today's network cannot differentiate between a user watching a random video clip from the web and an important customer being presented a demonstration video. With SFNet, high-priority applications can directly indicate the bandwidth required, so the network can make the bandwidth available.

To validate our implementation, we perform an experiment where a high-priority application (using TCP) runs in the presence of a low-priority UDP flow that is overwhelming the link. We create both flows using iperf, where the UDP flow is specified with a sending rate of 3 Mbps. The high-priority TCP flow submits a request for a bandwidth of 3 Mbps to SFNet.

Our results (Fig. 5) show that the TCP flow achieves an average rate of 280 Kbits/sec without reservation. By reserving bandwidth over SFNet, the TCP flow achieves 2.24 Mbits/sec—about 8 times more throughput. We also observe that the low priority UDP flow is also appropriately throttled (Fig. 5). We also observe that this has a dramatic effect on the quality of the playback of a streamed video. Such bandwidth reservation allows applications to express the tacit importance of each flow by explicitly reserving the appropriate network resources.

5. MULTICASTING SESSIONS FOR MULTI-WAY P2P CHAT

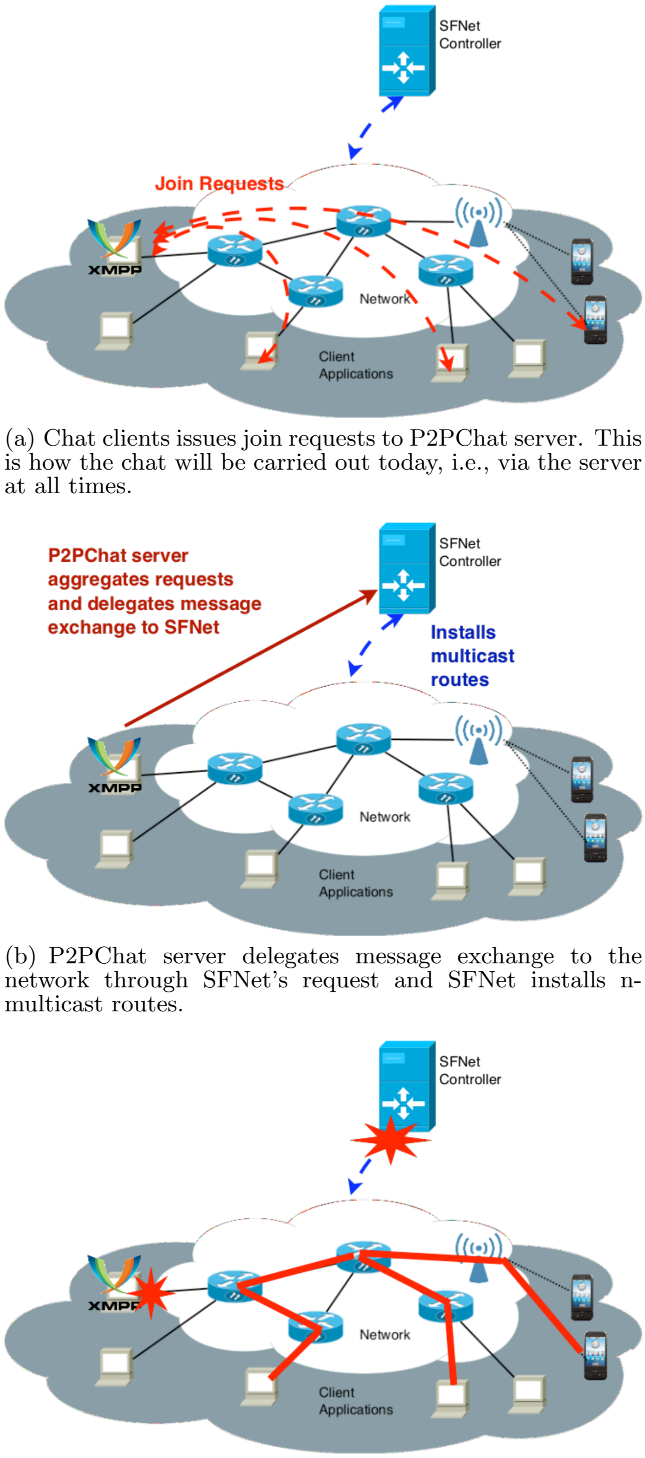
We now discuss how SFNet can support a multicast session for an application. Here, the application indicates to SFNet a set of IP addresses participating in the multicast with a selected multicast IP address; SFNet then returns a response (success or failure). Subsequently, messages sent to that multicast address will be delivered to the participants. SFNet finds the shortest network paths between the participants. Each message sent to the multicast IP address is duplicated where necessary in-network. To deliver packets among n participants, SFNet installs n multicast trees—from a host to the other $n-1$ hosts. Each multicast message is carried only once on each link of the multicast tree. This efficiency is critical for increasingly important telepresence applications such as high-definition multi-user video conferencing.

We next describe how we can use this multicasting in SFNet to improve the implementation of a chat service that uses XMPP as a rendezvous point to set up chat sessions, hereafter referred to as P2PChat. Fig. 6 describes a use scenario of P2PChat. To join the service, each user will submit a join request to P2PChat using the XMPP protocol (Fig. 6(a)). P2PChat aggregate requests for a chat session and submits the IP addresses of the participants in a request to SFNet, which installs the appropriate multicast routes for the session (Fig. 6(b)). The participant can then communicate with each other by sending messages to the multicast IP address. The chat messages are forwarded directly in-network, instead of going through the server (Fig. 6(c)).

Once a chat session is set up, the participants can communicate directly with each other using multicast sessions. This translates to reduced traffic as well lower latency. Our results show that by having participants communicate directly, we can reduce delay from 21 ms to 3 ms compared to communicating through the XMPP server residing in the same LAN. In the meantime, relieved of its message routing duty, the P2PChat server can scale to serve more users. Moreover, failures of the chat server will not affect any of the ongoing chat sessions. This opens up interesting possibilities, such as using a transient client in the network as the chat server.

6. RELATED WORK

Previous works, such as Resource ReSerVation Protocol (RSVP) [12] and Darwin [3], aims at developing application-aware networks. RSVP allows applications piggyback QoS request in their packets and request resources from routers the packets traverse. However, RSVP suffers from scalability and reliability problems and is therefore not widely supported. Darwin uses a global resource scheduler to handle



(a) Chat clients issues join requests to P2PChat server. This is how the chat will be carried out today, i.e., via the server at all times.

(b) P2PChat server delegates message exchange to the network through SFNet's request and SFNet installs n-multicast routes.

(c) Clients switch to P2P chat where the malfunction of the XMPP server and/or SFNet controller will not affect the chat session.

Figure 6: P2P chat via in-network mesh-casting (n-multicast)

requests from applications, and allocate network resources through "active networking", i.e., embedding code into application packets and executing the code in routers en route. However, "active networking" is plagued with concerns, such as security and computation loading. Also, the result of code execution might differ based on router's architecture, which is difficult to track.

Application developers has also try to seek solutions at the application layer. Numerous methods have been developed to estimate the end-to-end network capacity and route quality. Available bandwidth (ABW) estimation is one of the most well-studied techniques [7, 11, 14, 10]. Real-time multimedia applications often use the estimation to choose encoding rate [13]. Peer-to-peer applications uses ABW estimation for route selection, QoS verification, and traffic engineering over the overlay network [8]. However, due to the dynamic nature of Internet traffic, it is difficult to accurately estimate end-to-end available bandwidth [6]. Other than ABW, applications would also monitor the packet loss rate and apply different forward error correction mechanism. However, these techniques can only help applications remedy their performance loss, and not proactively improve it.

On the other hand, network operators are also trying very hard to understand the traffic from the applications. Traffic classification mechanisms are widely used to help network operators deploy appropriate QoS mechanisms, prevent network congestion, or to defend against network attacks. Many techniques can be used to classify application traffic from aggregated flows. Some are based on statistical properties of network traffic, such as packet size, inter-packet gap and flow duration [1, 4], while others are based on protocol signatures and deep packet inspection. None of these would be able to determine the importance of a flow with certainty.

With software-friendly networks, we are taking the "guesswork" out both from applications and networks, allowing the network to a service to the applications hence fulfilling their requirements directly. The central controller can also provide the appropriate authentication to relieve many security concerns [2].

7. CONCLUSION

In this short paper, we have presented our preliminary exploratory foray of how to provide software friendly network in the context of SDN. At the time of writing, SFNet has been deployed in our production network [16] allowing applications to make better use of our network. We are encouraged by how easy it is to provide conventional API like bandwidth reservation and multicasting, while supporting interesting new API like congestion enquiry.

What we presented is but a small step in the direction of creating software friendly networks, using a particular approach in an early prototype system. There are much more work to be done. We hope others will become similarly aspired towards the goal of creating software friendly networks, and explores the multitude of approaches possible in the solution space.

8. REFERENCES

[1] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli. Revealing skype traffic: when randomness plays with you. In *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 37–48, New York, NY, USA, 2007. ACM.

[2] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. In *Proc. ACM SIGCOMM Conference*, Kyoto, Japan, Aug. 2007.

[3] P. Chandra, Y.-H. Chu, A. Fisher, J. Gao, C. Kosak, T. Ng, P. Steenkiste, E. Takahashi, and H. Zhang. Darwin: customizable resource management for value-added network services. volume 15, pages 22 –35, jan/feb 2001.

[4] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli. Traffic classification through simple statistical fingerprinting. volume 37, pages 5–16, New York, NY, USA, 2007. ACM.

[5] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards and operating system for networks. In *ACM SIGCOMM CCR*, July 2008.

[6] C. D. Guerrero and M. A. Labrador. On the applicability of available bandwidth estimation techniques and tools. In *Computer Communications*, January 2010.

[7] M. Jain and C. Dovrolis. Pathload: A measurement tool for end-to-end available bandwidth. In *Passive and Active Measurement Workshop*, April 2002.

[8] M. Jain and C. Dovrolis. Path selection using available bandwidth estimation in overlay-based video streaming. *Comput. Netw.*, 52(12):2411–2418, 2008.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, 2008.

[10] B. Melander, M. Bjorkman, and P. Gunningberg. A new end-to-end probing and analysis method for estimating bandwidth bottlenecks. In *Global Telecommunications Conference, 2000. GLOBECOM '00. IEEE*, volume 1, pages 415 –420 vol.1, 2000.

[11] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell. pathchirp: Efficient available bandwidth estimation for network paths. In *Passive and Active Measurement Workshop*, April 2003.

[12] RFC 2205: The resource reservation protocol (rsvp). `http://tools.ietf.org/html/rfc2205`.

[13] Skype and network management. `http://share.skype.com/sites/en/2009/12/skype_and_network_management.html`.

[14] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *ACM IMC*, October 2003.

[15] D. L. Tennenhouse and D. J. Wetherall. Towards an active network architecture. *SIGCOMM Comput. Commun. Rev.*, 37(5):81–94, 2007.

[16] K.-K. Yap, M. Kobayashi, D. Underhill, S. Seetharaman, P. Kazemian, and N. McKeown. The stanford openroads deployment. In *WINTECH '09: Proceedings of the 4th ACM international workshop on Experimental evaluation and characterization*, pages 59–66, New York, NY, USA, 2009. ACM.