Demo: Application-Transparent Deployment of DTN via SmartNet

Lance A. Alt Justin P. Rohrer Geoffrey G. Xie
Department of Computer Science, Naval Postgraduate School
{laalt, jprohrer, xie}@nps.edu

ABSTRACT

In this paper, we present the SmartNet architecture, an open and extensible software framework for experimenting with and deploying application-transparent network adaptation solutions. The framework fashions a plugin-based system architecture where each plugin implements a small set of application or transport protocol specific network adaptation requirements and can be chained with other plugins to form a packet processing pipeline. Multiple concurrent packet pipelines are configurable to enable selected traffic flows to dynamically switch between native IP, split-TCP, and DTN, based on observed network conditions. The end user is able to configure not only which flows should use split-TCP or DTN, but also the conditions under which each option should be used.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*store-and-forward networks, network topology*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*bundling protocol, split-TCP*; C.2.6 [**Computer-Communication Networks**]: Internetworking—*routers*

Keywords

Disruption-tolerant networks; transparent gateways; middleboxes; split-TCP

1. INTRODUCTION

The predominant approach to integrating non-IP and IP networks follows a vertical overlay model. In the case of Disruption Tolerant Networking (DTN), it is either IP-over-DTN or DTN-over-IP. This layered approach is simple to design and implement, by requiring no data translation or tracking of application state inside the network. However, it not only introduces extra encapsulation overhead, but more importantly, imposes *least-common denominator* semantics when moving data across network boundaries, and as such, may severely degrade the performance of many applications originally designed to work over end-to-end IP connectivity.

Recently, we proposed an alternative approach to integrating IP and DTN that can increase application performance over DTN while requiring no code changes to either applications or the host protocol stack. Central to this "application-transparent" approach [5] is the introduction of a proxy gateway that (i) splices the IP and DTN functionalities into a single logical network layer, and (ii) supports application-specific adaptation and notification. The system proposed normally routed traffic over native IP and switched only to DTN for specific traffic classes when link disruptions were detected. It also included mitigation for application-level timeouts, e.g. inserting keep-alive messages on behalf of a remote SIP chat client [5] to boost application performance and notify the end user of delays.

In this work, we present the SmartNet architecture, an open and extensible software framework for experimenting with and deploying application-transparent network adaptation solutions such as our previously proposed DTN proxy gateway. The framework fashions a plugin-based system architecture where each plugin implements a small set of application or transport protocol specific network adaptation requirements and can be chained with other plugins to form a packet processing pipeline. In our initial implementation, multiple concurrent packet pipelines can be configured to enable designated packet flows to dynamically switch between native IP, split-TCP, and DTN, based on observed network conditions. The end user is able to configure not only which flows should use split-TCP or DTN, but also the conditions under which each option should be used. An advanced user is provided with an API to enable the creation of new plugins that can form new elements of the packet pipelines through standard input and output interfaces, complete with support for new applications, and new configuration options.

2. SMARTNET DESIGN

The challenges facing the deployment of DTN are not unique. While there are abundant scientifically vetted proposals for network enhancements, many computers and mobile devices still run the basic stack developed more than a decade ago. Commercial products such as WAN optimizers and other kinds of middleboxes are mostly proprietary, too bulky and too costly to be widely adopted. In designing SmartNet, we seek to provide a *general software solution* for deploying network optimization.

In this section, we first state our primary objective and a set of desirable system properties dictated by it. We then describe a "divide and conquer" methodology to further modularize the development and deployment of network optimization software. Finally, we detail several important elements of our current SmartNet design, including the software architecture, the major functional building blocks, and the key steps taken to address potential performance bottlenecks.

2.1 Primary Objective and System Properties

We envision the users of the SmartNet gateway software to include both the developers of network optimization solutions and the network operators who deploy such solutions. For both groups of users, which constitute the core of the network optimization ecosystem, a top concern is how easily an optimization solution can be integrated into an existing network. Therefore, our primary objective is to support rapid deployment of all sorts of network optimizations in one network. To this end, we view the following characteristics as essential for SmartNet.

- **Application-transparent**. SmartNet should boost the network performance of applications without requiring code changes to either the applications or the host OS stack. Furthermore, it should provide means for explicit notifications of network optimist and prevent protocol or application timeouts.
- **Open and extensible**. SmartNet should provide standard public interfaces for (i) adding new network optimization modules (either source or compiled code), and (ii) enabling a specific subset of optimization modules and configuring their trigger conditions.
- **Per-flow optimization**. SmartNet should support concurrent executions of optimization logic for different packet flows (application-specific, destination specific or other granularities).

Clearly, there will be a performance trade-off for providing the level of flexibility and abstraction described above. It is important for SmartNet to control this cost to a level that does not significantly impact the performance gains of the deployed network optimization solutions. After all, performance is the reason why operators deploy network optimizations in the first place.

2.2 Plugin-based System Architecture

Our design is focused around a dynamic plugin based architecture. A central core manages all plugins and certain common functionality such as the central storage facility (CSF). Plugins are distinguished broadly into two classes: "optimization" plugins process packets and perform the actual network optimization while "meter" plugins monitor and provide updates of network state information required for determining which kind of optimization, if at all, should be performed for each user flow.

We have created and evaluated several plugins specific to providing integrated IP and DTN transport to SIP/UDP and HTTP traffic. Since we have presented the integration of DTN and SIP/UDP in a prior paper [5], we will focus on how SmartNet supports HTTP flows in this paper. More specifically, we have designed a SplitTCP plugin to minimize the negative performance impact of a network layer disruption to TCP connections, including those carrying HTTP traffic. The same plugin needs to run on both end points of a Web session. Together they create three connections for delivering the packets: a TCP connection between the client and the client-side SmartNet, a TCP connection between the server and the server-side SmartNet, and a TCP or DTN connection between the two SmartNets. This way, the plugins can dynamically switch between the normal TCP/IP transport mode and DTN depending on the network condition, while preventing the TCP connection on either the client or server side from retransmitting unnecessarily or timing out when a network disruption occurs.

2.3 Major Functional Building Blocks

Our SmartNet implementation consists of several distinct and modular components: Packet pipelines, asynchronous messaging system, and composable triggers.

2.3.1 *Packet Pipeline*

The key building block in the SmartNet is the notion of a packet pipeline which provide network optimization for one user flow at different granularities (e.g., per destination address or subnet and per application). The pipeline consists of one or more plugins connected to create a directed acyclic graph. Multiple parallel pipelines can coexist and run at the same time. More specifically, packets of different user flows are de-multiplexed onto different paths within the gateway based on configurable criteria such as their application type and destination addresses.

Individual plugins are connected via packet queues. Each plugin contains one and only one input queue. Utilizing queues as the input for each plugin allows the SmartNet to be resilient to plugins that process packets in non-constant time. A fast processing plugin can continue to process even if its destination plugin contains a backlog of packets. Furthermore, the fast plugin can query the size of the backlog and choose to temporarily route packets to a different destination until the backlog clears sufficiently.

A plugin may include a decision point (with a configurable decision criteria or threshold) that causes it to output to more than one queue, resulting in a fork in that pipeline. Also, the plugins and queues may be reused across multiple pipelines, resulting in a topology that is a mesh or lattice, not a series of discrete linear pipelines.

Figure 1 shows an example of a complex plugin pipeline in a lattice configuration. Decision points within each plugin can cause forks in the pipeline, dynamically sending packets on different paths. Plugin developers can reuse existing publish/subscribe parameters as input to their decision points as appropriate. Merging of two paths is also supported allowing for selected packets to make small deviations for additional processing before rejoining another pipeline. Such a configuration can be customized on a per deployment basis involving a confederation of SmartNet gateways.

During testing, we have constructed a variety of complex piplines that perform a variety of tasks such as: Network Address Translation, protocol-based packet classification, packet aggregation, and MTU enforcement. Preliminary testing shows that the overhead of using long pipelines is minimal, allowing network administrators the ability to replace numerous single-purpose middleboxes with a single SmartNet solution.

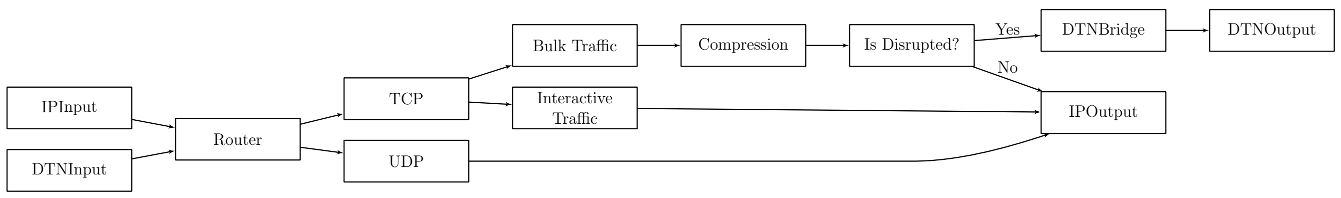

Figure 1: An example of a complex processing pipeline. SmartNet is capable of arranging plugins in a mesh or lattice configuration allowing unlimited processing flexibility.

2.3.2 *Pub-Sub Asynchronous Messaging*

Plugins from different developers may need to exchange control messages in a SmartNet configuration. For example, a meter plugin and an optimization plugin may need to communicate in order for the latter to receive timely updates of certain network state information. To accomplish this objective, the SmartNet provides a general purpose publish-subscribe (pub-sub) messaging system.

The messaging system is dynamic, allowing plugins to subscribe to information feeds either at initialization or on an as needed basis. Plugins can unsubscribe when certain information is no longer relevant, preventing unnecessary message processing. Incoming messages are asynchronously placed in the receiving plugin's message queue, allowing the plugin to process the message when convenient.

For the publisher, all messages are centrally managed by the SmartNet core. The publishing plugin is not concerned with who has subscribed to a feed. Once the plugin has pushed the message to the pub-sub subsystem, it can return to its normal processing without delay. In addition, a publisher can be configured to respond to specific requests for information. This "pull" method is useful for messages that may be costly to compute and are needed infrequently.

2.3.3 *Composable Triggers*

By utilizing the asynchronous messaging system, SmartNet provides a robust interface for network operators to create composable triggers. Messages from multiple independent meter plugins can be utilized to make optimization decisions at any point in the packet pipeline. Triggers such as "packet-loss rate $\leq 1\%$ **AND** bandwidth-$\times$-delay product < 400 Kb" can easily be composed.

For example, lets assume we have a router which provides load balancing over two external connections. A simple load balancer would attempt to divide the packet flows evenly over the two connections which works well assuming the external connections are reliable. Now assume that both connections are wireless and suffer from variable latency and loss rates. The simple load balancer would operate ineffectively since it is blind to the reliability of the external connections.

The load balancer functionality can easily be written to operate as an optimization plugin on the SmartNet. Likewise, the two external connections would each be measured by a plugin or plugins providing meter functionality. The meter plugin(s) export information such as the packet loss and bandwidth-delay product, which in turn is subscribed to by the load balancer. With this information, the load balancer can now effectively balance the packet load, taking in to account the reliability of the external connections.

3. EVALUATION

To test the SmartNet design, we have created a Linux based implementation written in C++ and consisting of both the core SmartNet components and several plugins as described in the previous section. The plugins are compiled separately as Linux shared libraries, allowing SmartNet to dynamically load and configure plugins at runtime. The pipeline, as well as any plugin specific configuration parameters, is configured using text based configuration files.

Included in our initial release are the IPInput and IPOutput plugins which read and write packets from the Linux networking stack, the DTNInput and DTNOutput plugins which read and write packets from BBN Technologies' Spindle implementation [4], and the DTNBridge plugin which adapts TCP packets to operate over a DTN-based protocol. Several other plugins, such as a routing plugin and SplitTCP plugin, are included in the disribution. The open source code is source and available on our website[1].

SmartNet is designed to run in user-space, receiving packets from the Linux kernel via the Netfilter NFQUEUE target. This allows system administrators to selectively redirect traffic to the SmartNet using firewall rules. Our implementation uses `libcrafter` [2] to perform all basic packet parsing and manipulation routines. We use the lwIP networking stack [1] in the DTNBridge plugin to simulate the remote host. This allows us to efficiently strip out TCP flow control and unnecessary header information when delivering packets over DTN. Likewise, on the remote end lwIP emulates the sending host to repack the DTN data in to a TCP packet.

The full results of our testing and evaluation may be found in the Master's Thesis of Lance Alt [3].

4. REFERENCES

[1] lwIP - A Lightweight TCP/IP stack, 2012. `http://savannah.nongnu.org/projects/lwip`.

[2] libcrafter, 2014. `https://code.google.com/p/libcrafter`.

[3] L. A. Alt. Application transparent HTTP over a disruption tolerant SmartNet. Master's thesis, Naval Postgraduate School, Monterey, CA, USA, September 2014.

[4] R. Krishnan *et al.* The SPINDLE disruption-tolerant networking system. In *Proceedings of the IEEE Military Communications Conference (MILCOM)*, Orlando, FL, USA, 2007.

[5] J. P. Rohrer and G. G. Xie. DTN gateway architecture for partially disconnected telemetry environments. In *Proceedings of International Conference on Connected Vehicles and Expo*, Las Vegas, NV, USA, Dec. 2013.

[1]`https://github.com/lancealt/npsgate`