MultiFlowDispatcher and TCPSpeaker

Harald Schiöberg, Daniel Levin

Deutsche Telekom Laboratories
An-Institut der Technischen Universität Berlin
{harald,dlevin}@net.t-labs.tu-berlin.de

1 Introduction

TCP performance in multihop wireless is known to decay almost exponentially with the number of traversed wireless hops. We aim to overcome this by not running TCP from the wireless network altogether. We propose a Click based architecture to provide a generic proxy that allows for arbitrary transport protocols within the mesh while retaining full TCP compatibility to hosts outside our network.

For this purpose we developed the abstract MultiFlowDispatcher element class, which handles per-flow state within click in a generic way. This is then used to implement the TCPSpeaker, a fully functional port of the 4.4BSD-lite TCP stack to click.

As our goal is to run a multihop network without TCP, TCPSpeaker is designed to be coupled with future implementations of wireless-friendly transport protocols, which will also be implementations of the MultiFlowDispatcher. This will form the architecture of Figure (a): mesh nodes terminate TCP connections at the edge, and transfer data using arbitrary implementations of the MultiFlowDispatcher to retain fully transparent TCP functionality towards the outside of the network. Nodes A and B are serving clients with a few open TCP connections, or might alternately act as Internet gateway, proxying the in- and outbound Internet traffic.

2 Architecture

Figure (b) illustrates the new abstract Element MultiFlowDispatcher. It spawns one MultiFlowHandler per flow. Implementations of MultiFlowDispatcher also implement MultiFlowHandler, e.g. TCPSpeaker implements MultiFlowDispatcher and TCPConnection implements MultiFlowHandler.

The core functionality of MultiFlowDispatcher is to manage the instances of MultiFlowHandlers, one per bi-directional flow. Upon packet arrival at one of the push ports of the dispatcher, a lookup in a HashTable is used to dispatch the packet to the corresponding handler. Calls to pull are currently distributed round-robin among the handlers. Handlers then directly call the push and pull methods of Elements connected to the dispatcher.

MultiFlowDispatchers have either one input and one output port for unidirectional flows, or two input/output ports in the bidirectional case (TCPSpeaker is an example of a bidirectional implementation of MultiFlowDispatcher). For bidirectional Elements, port processing should be consistent: Port 0’s output processing should be the same as port 1’s input processing and vice versa.

TCPSpeaker (Figure (c)) implements the MultiFlowDispatcher. It merely tracks the TCP connections, while the actual per-connection TCP state machine is implemented in the TCPConnection class.

TCPSpeaker can also be seen as having two sides: input 0 and output 1 form the stateful side, which is usually connected to some remote host running TCP outside our network.

On the stateless side output 1 emits all TCP data as a complete in-order bytestream. (Actually the payload segments are emitted mostly unchanged, only put in correct order and truncated in case of overlap, TCP options are removed) No pure control segments (any segments not containing payload) are emitted. Similarly input 1 expects packets containing TCP and IP headers, but only the IP addresses and TCP ports are actually used for processing, all other fields are ignored. The data to the stateless input must be delivered in-order and without any gaps.

The intended use is to couple TCPSpeaker with other MultiFlowDispatcher realizations to build a Layer 4 protocol translator. As a simple example, by directly connecting the stateless sides of two TCPSpeakers, a completely operational TCP split proxy can be built.

3 Future Work

We are currently actively pursuing the issue of how to propagate flow- and congestion control information, which is partly handled by the pull property of the stateless side, but requires smarter scheduling of the incoming pull calls. (We are investigating directly connecting the Handlers without multiplexing through the Element port) Extending the architecture with support for client mobility is also current work in progress, which mandates a serialization infrastructure for MultiFlowHandler. Currently the MultiFlowHandler is not derived from Element, yet it implements the push and pull methods. It may be feasible to implement a realization of MultiFlowDispatcher, that can spawn actual Elements to add multi flow support to any existing Element.