Network coding with traffic engineering

Wenjun Hu  Miguel Castro  Jon Crowcroft  Greg O’Shea  Antony Rowstron
wh214@cl.cam.ac.uk  mcastro@microsoft.com  jac22@cl.cam.ac.uk  gregos@microsoft.com  antr@microsoft.com

1. INTRODUCTION

In network coding, a router in the network mixes information from different flows. In the seminal work by Ahlswede et al [1], network coding is established as a technique to potentially increase the network capacity.

Motivated by these results, the Opportunistic Coding Protocol (COPE) [6, 7] applies network coding to wireless mesh networks, which operate on a broadcast medium and are constrained by its shared nature. Experimental results from a real testbed confirm that COPE has practical benefits. We note, however, that the network coding schemes proposed so far in the literature, COPE included, are generally sensitive to flow topologies. This is because encoding and decoding take place on different nodes inside the network, and the decoding phase often requires side information.

Therefore, we explore the use of traffic engineering to achieve higher coding benefits. Our scheme aims to compute routes to create favourable flow topologies for the coding protocol, to improve the overall throughput. In the following sections, we first describe the standalone routing and coding protocols and outline our proposed traffic engineering scheme.

2. CODING PROTOCOL

COPE includes opportunistic listening, where nodes snoop on the medium, and opportunistic coding, where relay nodes perform coding as opportunities arise. It is best illustrated by the typical scenario shown in Figure 1. A wishes to send a packet p to C via relay R, while B wishes to send a packet q to D via R. Meanwhile, D and C could overhear p and q respectively. In current forwarding approaches, the relay can only send one packet at a time, and would require two transmissions to deliver both p and q. In COPE, however, the relay R realises the situation, encodes the two packets together using bit-wise XORing, and broadcasts the XORed version. By XORing the received encoded packet with their respective overheard packet, C and D could then retrieve the other packet. The scheme allows the relay to save transmissions, one in this case, thus improving network throughput when compared with current forwarding approaches without coding.

3. ROUTING PROTOCOL

We choose Virtual Ring Routing (VRR) [3] as our routing protocol. VRR is inspired by overlay routing algorithms in Distributed Hash Tables (DHTs), but implemented directly on top of the link layer. It uses random unsigned integers to identify nodes, and organises the nodes into a virtual ring in order of increasing identifier. Each node maintains a virtual neighbour set, vset, and a physical neighbour set, pset. The vset, of cardinality r, contains the node identifiers of the r/2 closest neighbours clockwise and anti-clockwise in the virtual ring. The pset contains the identifiers of nodes that it can reach at the link layer. VRR sets up and maintains routing paths between a node and each of its virtual neighbours. Nodes route packets to destination identifiers by selecting, from among all endpoints in its routing table, the endpoint numerically closest to the packets destination. The packet is then forwarded to the physical neighbour which makes progress towards the selected endpoint. (Figure 2)

4. TRAFFIC ENGINEERING

Figure 3 illustrates the potential benefits of traffic engineering. There are no coding opportunities in this scenario but this can be improved. By propagating appropriate information within a 2-hop radius, B can determine that R is a physical neighbour of D, that D can overhear packets sent by A, and that C can overhear packets sent by B. Therefore, B can route packets to D via relay R to enable coding at R, which could improve the overall throughput as illustrated in Figure 1.

To enable this form of local traffic engineering, a routing protocol should provide the flexibility to adjust paths locally on a per-packet basis, while ensuring that the modified paths converge to the final packet destinations and minimising any
Packet forwarding in VRR involves two stages: first a node selects the path endpoint towards the final destination and then the next hop towards that endpoint. In some cases, there are multiple physical neighbours that can be used to reach the same endpoint. Currently, VRR uses link quality metrics (ETT [5]) to select the next hop from among these neighbours. However, it is possible to adjust the metric to choose a next hop that maximizes coding opportunities. Additionally, it is possible to relax the criteria for the choice of the destination endpoint, by selecting an endpoint which is not the numerically closest, but is still numerically closer than the current node. This will ensure that packets are still delivered correctly, while providing more flexibility in the choice of next hop to increase coding opportunities.

We could refine the ETT metric to take into account coding opportunities. Each node maintains a count of packets sent to each physical neighbour in the last T seconds, and distributes the information to its physical neighbours using VRR’s hello messages. This information allows nodes to estimate the probability that their physical neighbours can perform coding when relaying packets. If a packet is likely to be encoded at the next hop, its transmission time in the following hop is effectively reduced, because coding increases the perceived bandwidth. The coding probability can be used to adjust the ETT metric for the next-hop selection to take this coding effect into account.

5. IMPLEMENTATION

For VRR, the Windows implementation is based on the Mesh Connectivity Layer (MCL), which is a virtual layer between the link layer and the IP layer, and appears as a virtual link to an unmodified TCP/IP stack [4].

COPE was designed to be independent of upper and lower layer protocols as far as possible. It was originally implemented as an extension on the forwarding path in the Roofnet software[2], approximately as a shim between the MAC and routing layers. We reimplemented COPE as a sublayer of the MCL driver in the Windows kernel, incorporating coding related functions at the appropriate packet processing points. Currently we are implementing local traffic engineering.

6. REFERENCES