NUROP CONGRESS PAPER
ENHANCING, AND IMPLEMENTING THE SNOOP PROTOCOL IN LINUX

INDRADEEP BISWAS\textsuperscript{1} AND A.L. ANANDA\textsuperscript{2}

School of Computing, National University of Singapore
3 Science Drive 2, Singapore 117543

ABSTRACT

Due to the shortcomings of TCP over wireless networks, Performance Enhancement Proxies (PEP) are important in ensuring better utilization of network resources. The PEP we dwell on in this paper is known as the Snoop Protocol, a link layer - TCP aware mechanism, which performs recovery for lost packets on the last mile wireless local loop.

Since the Linux user-base has been growing substantially over the recent past, the Snoop implementation for Linux can create benefit for a bigger community. It can also serve as a reference to those in research to benchmark and implement new ideas in the area of PEP.

Performance tests show that the Snoop module for Linux performs consistently better than ordinary TCP over wireless, and its average case performance is on par with the BSD implementation.

A scheme is proposed to enhance Snoop’s capability in handling short disconnections. A Slow-Acknowledgement feature, if added to Snoop, would use three duplicate acknowledgements and TCP’s congestion avoidance mechanism to smooth out the throughput drop during short disconnections.

1. INTRODUCTION

TCP, the most widely used transport layer protocol, was designed and optimized for wired networks. When TCP is used over wireless networks, the high bit error rates, and frequent small disconnections, cause serious degradation to its performance.

Since it is unreasonable make changes to the core TCP protocol, efforts are being made to work around its shortcomings. The Snoop Protocol [1] is a TCP-aware, link-layer recovery scheme, which increases TCP’s performance over wireless by hiding the packet losses over the wireless hop.

Snoop has been implemented only on the BSD operating system. Since BSD is not popular, the use of Snoop is limited primarily to the research community. In order for the growing number of users of the Linux platform to benefit from the enhancement Snoop offers, it would be useful to have an implementation of Snoop in Linux. The researchers developing ideas for other Performance Enhancement Proxies (PEP) can use Snoop not only for benchmarking the performance of their ideas, but also as a framework to implement their own PEP.

\textsuperscript{1} Student
\textsuperscript{2} Supervisor
2. BACKGROUND

In wireless networks, packet losses occur in two different scenarios: bit-level errors and short disconnections. Bit-level errors often arise when packets, being transmitted over the wireless hop, are influenced by high energy interference. These corrupted packets are silently discarded by receiver, causing packet loss. TCP sender learns of the packet loss through three duplicate-acknowledgements (dupacks), or a retransmission timeout. When three dupacks are received, lost packets are retransmitted, and the congestion window is halved. In case of a retransmission timeout, TCP invokes slow start, i.e. drops the window size to one. Although these mechanisms were designed to stop TCP from pushing in excess data into a congested network, they severely reduce the throughput of the connection even when the reason for packet loss is not congestion. Even though the full bandwidth is available, the pipe is not fully utilized because the intermittent losses give TCP an impression of congestion.

3. THE SNOOP PROTOCOL AND ITS IMPLEMENTATION

The Snoop Protocol was designed to eliminate the drop in throughput due to intermittent packet loss over wireless. It shields the sender from the vagaries of the wireless link, without sacrificing the end-to-end semantics, or requiring any changes to the existing implementations of TCP. The Snoop concept is based on a set of simple policies on handling the packets that traverse through the last hop router before wireless [2].

Snoop maintains a cache of all unacknowledged TCP packets sent from the Fixed Host (FH) to the Mobile Host (MH). The Snoop agent contains a chain of connections, and each connection contains a chain of packet slots for buffering. When a new packet arrives from FH, the Snoop agent adds a copy of the packet to the chain of packet slots for the connection to which it belongs to. [Figure 1]

When an acknowledgement is received, the BS clears the acknowledged packets from its buffer. When a dupack is received, Snoop assumes packet loss over the wireless link and retransmits the lost packet from its cache. It then drops the dupack, effectively shielding the fixed host from the packet loss over the wireless hop [Figure 2].
NUROP – Enhancing and Implementing the Snoop Protocol in Linux

While caching each packet from the FH, a timer is added to the buffer. If the packet is not acknowledged within twice the RTO of the wireless hop, Snoop assumes a packet loss, and performs a local recovery.

The Snoop is implemented as a Linux kernel module. The module gains access to the TCP packets using the kernel IP_FORWARD hook made available by the Netfilter framework. Since the Netfilter framework is bundled with the kernel distribution, Snoop module can be used on any BS running Linux without requiring any modifications to the kernel.

4. PERFORMANCE TESTS AND BENCHMARKS

The tests have been conducted with a 10 Mbps Ethernet connection between FH to BH and 2 Mbps wireless connection between BH to MH. A delay of 200 ms is used whenever constant delay applies.

![Figure 3. [a] Delay between BS and MH vs. Throughput (left); [b] Delay between FH and BS vs. Throughput (centre); [c] % of packets dropped vs. Throughput (right)](image)

We find that even with variable delay between BS and MH, Snoop performs consistently better, with a peak performance of twice as that of normal TCP [Figure 3 a]. With variable delay between FS and BH, the peak performance is reached when the delay was about 400ms. This is because 400ms allows the Snoop agent adequate time to perform local recovery when the wireless losses occur. In this test, we see that Snoop gives a peak performance of about three times that of normal TCP [Figure 3 b].

With 2% corruption along the wireless link, the peak performance of twice the throughput of TCP is reached. The amount of performance boost with snoop drops with higher bit corruption rates, because the delay on the links do not allow snoop with enough time to attempt multiple local recoveries [Figure 3 c].

5. ENHANCING SNOOP WITH SLOW-ACK (SLACK) SCHEME

In the case of short disconnections, Snoop’s recovery attempts fail until the mobile host finds the connection again. If the disconnection is long enough, TCP faces a retransmission timeout and goes into slow start, which significantly reduces the throughput [Figure 4].

We know that the congestion control invoked by three dupacks drops the congestion window by half, which is a more lenient than retransmission timeouts. Hence, in case of a short disconnection on the wireless hop, the snoop agent can use a Slow-Ack (SLACK) – the last ack propagated to the Fixed Host to force the TCP sender to go into congestion avoidance than slow start. This causes the congestion window to reduce slowly, than in one shot [Figure 5].

Since most short disconnections recover soon after a single RTO timeout, with the application of SLACK, the performance drop can be reduced by half. In case of big fat pipes,
the window size is generally very large. If SLACK is used, even after multiple timeouts, the final congestion window size at the sender will be much greater than one when connection resumes. Hence, we can expect significant performance improvement of Snoop with the SLACK feature.

6. CONCLUSION

The Snoop protocol on Linux has been carefully implemented to enable it to run on any Linux based system without requiring any modification to the kernel. It has been successfully benchmarked to confirm the performance enhancement to TCP. Not only does Snoop on Linux corroborate the concept of the protocol, it also shows that the performance of the implementation on Linux is on par with the Snoop’s BSD implementation.

A solution – Slow-Ack for Snoop is proposed to enhance Snoop’s capability of handling short disconnections. The scheme proposes to increase performance by progressively dropping the congestion window before TCP sender gets to time out, rather than having the sender invoke slow start. Further work can be done to develop and implement this idea, since a working Snoop module in place.

7. REFERENCE