An Implementation and Experimental Study of the Variable-Structure Congestion Control Protocol (VCP)

Xiaolong Li Homayoun Yousefi'zadeh
Department of EECS
University of California, Irvine
[xiaolonl,hyousefi]@uci.edu

***Abstract*— The Variable-structure Congestion control Protocol (VCP) has been proposed as an alternative approach to the eXplicit Control Protocol (XCP). In our earlier work, we reported the NS2 simulation results of cross-layer profiling studies of VCP, XCP, and TCP+REM in encrypted wireless networks. Our studies utilized finite-state Markov chains to model bit error characteristics of wireless links and applied per packet link layer FEC codes in order to compensate for such errors. Our simulation results showed that VCP takes a significant step toward addressing the tradeoff between the performance and practicality of implementation.**

In this paper, we report the results of our implementation of VCP in the Linux kernel. Our implementation of VCP consists of a number of Linux Loadable Kernel Modules (LKMs) associated with transmitting, intermediate, and receiving nodes. Our implementation allows for the co-existence of VCP with standard transport protocols such as TCP and UDP. Utilizing an experimental wired testbed capable of realistically emulating the fading characteristics of wireless links, we profile the performance of the Linux implementation of VCP. Based on our profiling results in the Linux kernel, we observe (1) the need for protecting protocol's metadata as well as data against bit errors, and (ii) that VCP represents a high performing yet practical congestion control protocol for encrypted wireless networks.

I. Introduction

A significant volume of research [6], [13], [16], [5], [1] has been conducted by the research community in improving the efficiency and fairness of TCP congestion control algorithms [8]. However, the prevalence of high Bandwidth-Delay Product (BDP) networks creates significant challenges for such congestion control schemes. To that end, various mechanisms [21], [17], [4] have been proposed to either adaptively adjust sending window size by amending the parameters of Additive-Increase Multiplicative-Decrease (AIMD) [3], use different congestion signals [14], [12], [10], [2], or even explicitly signal the congestion information to the sender [19], [9]. Often times, such mechanisms fail to simultaneously achieve high utilization and fairness while maintaining low persistent queue length and minimizing congestion-induced packet drop rate.

In contrast, XCP [11] and VCP [20] decouple the fairness control from the efficiency. While XCP uses Multiplicative-Increase Multiplicative-Decrease (MIMD) for efficiency and AIMD for fairness control, VCP defines three levels of congestion and does Multiplicative-Increase Additive-Increase Multiplicative-Decrease (MIAIMD) in three regions of congestion, respectively. By encapsulating congestion related information into the header of packets, both protocols achieve high utilization, low persistent queue length, insignificant packet loss rate, and sound fairness. Unfortunately, XCP requires multiple bits in packets header introducing significant deployment obstacles. On the contrary, VCP is able to achieve a performance comparable to XCP using two ECN bits while keeping compatibility with a variety of existing protocols.

More importantly, VCP stacks up more favorably than XCP since the manipulation of multiple bits in the IP packet header by a protocol is subject to major practicality implications in encrypted networks. For example, High Assurance IP Encryption (HAIPE) standard [25] only allows for bypassing of six Differentiated Services (DiffServ) bits and two ECN bits across its encryption boundary.

Our earlier works of [22] and [23] reported the NS2 [24] simulation results of cross-layer profiling studies of VCP, XCP, and TCP+REM in encrypted wireless networks. They utilized finite-state Markov chains to model bit error characteristics of wireless links and applied per packet link layer Forward Error Correction (FEC) codes to compensate for such errors. The simulation results showed that VCP takes a significant step toward addressing the tradeoff between the performance and practicality of implementation.

In this paper, we report the results of our implementation of VCP in Linux and construct a testbed to conduct an experimental study. Specifically, the contributions of this paper include:

- *VCP implementation:* To our best knowledge, this work is the first implementation of VCP. Our implementa-

This work was sponsored by grant no. COM06-10223 from Boeing Integrated Defense Systems and UC Discovery Industry-University Cooperative Research Program.

tion consists of a number of LKMs associated with transmitting, intermediate, and receiving nodes of VCP. Most importantly, our implementation allows for the co-existence of VCP with standard transport protocols such as TCP and UDP.
- *Protocol evaluation:* Utilizing an experimental wired testbed of Linux nodes capable of realistically emulating the fading characteristics of wireless links, we profile the performance of the Linux implementation of VCP. The paper demonstrates (i) the need for protecting protocol's metadata as well as data against bit errors, and (ii) that VCP represents a high performing yet practical congestion control protocol for encrypted wireless networks.

The rest of the paper is organized as follows. In Section II, we present the fundamentals of VCP. Section III presents our implementation approach. Experimental studies are presented in Section IV. Finally, we present several conclusions and discuss future work in Section V.

II. Background

In this section, we first review the fundamentals of VCP. Then, we present four principles of our implementation of VCP in Linux.

A. Fundamentals of VCP

Fundamentally, VCP remains a window-based protocol and is designed to regulate the $cwnd$ with different congestion control policies according to the level of congestion in the network. VCP defines three levels of congestion: low-load, high-load, and overload allowing for encoding the level of congestion into two ECN bits in the IP packet header. Upon arrival of a packet, each VCP-capable router does: (i) compute the load factor of each of its links and map it to one of the congestion levels, and (ii) update the ECN bits in the packet header. A more congested downstream router can further change the level of congestion by overwriting it. Finally, the receiver signals the sender with the congestion information via acknowledgment (ACK) packets. Consequently, VCP applies three congestion control policies: MI in the low-load region, AI in the high-load region, and MD in the overload region. The MI region is utilized to eliminate the slow start characteristic of TCP while the AI and MD regions preserve the fairness characteristics of TCP.

B. Principles of Implementation

In contrast to XCP, VCP is desirable in terms of deployment. Our implementation dedicates to highlight the strengths of VCP and thus, four design principles are defined:

- *Be transparent to applications:* The protocol should not require any change to applications, which is important for a transparent deployment. Namely, all TCP-based applications (e.g. FTP, HTTP) are able to communicate via VCP without being aware of the existence of VCP.
- *Offer easy integration in Linux:* VCP should be implemented as a LKM. Linux LKMs allow for adding new kernel features without recompiling the existing kernel, which means that a VCP module could be compiled, loaded, and unloaded without rebooting the system.
- *Preserve backward compatibility:* The implementation should be entirely compatible with existing TCP options and schemes, such as fast-recovery, retransmission, etc.
- *Be friendly to all transport protocols:* A VCP-capable router should be capable of dealing with other standard transport protocols such as TCP and UDP. More importantly, such friendliness must not adversely affect the performance of VCP.

III. Linux Implementation of VCP

A. The Overall Architecture

Many congestion control protocols including VCP operate by manipulating the $cwnd$ of the sender. To expedite the development, we take an implementation approach that treats VCP as a protocol independent of TCP, while taking advantage of its important features such as fast recovery and retransmission. More specifically, VCP is implemented as a "layer 3.5" protocol between the IP and the transport layer. From the view point of the end nodes, it appears to be a "dummy" layer while transmitting. The dummy property comes from the fact that there is no new protocol header introduced for VCP.

Furthermore, by directly grafting TCP on top of VCP, the overhead introduced by crossing protocol layers is minimized. The latter will be discussed in more detail in section III. Under such architecture, for TCP-based applications, TCP remains the underlying transport mechanism that delivers data except that the congestion control is taken over by VCP. Thus, there is no need to change existing TCP-based applications for VCP deployment. Put simply, our design dedicates to enable VCP functionality with minimum overhead, while keeping compatibility with legacy TCP stacks.

Before describing the implementation in more detail, we provide an example scenario to explain the operation of VCP. Assuming the establishment of an FTP session, the flow of events in our example is as follows:

1) As operated normally, the FTP server renders FTP data to TCP stack, whereby TCP packets are created. Instead of being passed to IP layer, TCP packets are detoured to VCP.

2) VCP does nothing but (i) faking the packets as they are coming from VCP, (ii) forwarding the modified packets to IP, and (iii) marking ECN bits as "01" denoting the "LOW_LOAD" in the context of VCP.
3) IP processes packets normally, i.e., it encapsulates them with their IP headers, fills the protocol field of IP header with the number of the transport layer protocol associated with packets, and transmits them out. Note that the transport protocol id has been changed from TCP to VCP in the last step.
4) A VCP-capable router receiving packets takes the following actions. First, the router examines the ECN bits in a packet header and compares them with latest congestion level of the outgoing link of the packet. If more congested, the router updates the ECN bits with "HIGH_LOAD" or "OVER_LOAD". Otherwise, the packet is simply forwarded onwards. Note that the ECN bits could be updated by all VCP-capable routers along the path from the sender to the receiver.
5) On the receiving side, VCP accepts packets and processes each packet as follows. First, the congestion information in ECN bits is saved. Then the packet is forwarded to the TCP stack. When sending an ACK packet back, VCP reads the saved congestion information and marks the ECN bits in the header of the ACK correspondingly.
6) A VCP ACK received at the sender is converted back to a normal TCP ACK packet and delivered to the TCP stack after saving its ECN bits. Meanwhile, VCP adjusts the $cwnd$ based on the updated congestion information.

B. Linux Congestion Control Architecture

Since VCP contributes by manipulating $cwnd$ solely based on the router feedback, it may potentially replace the entire Linux congestion control scheme. However, Linux performs congestion control in our implementation based on the reasons discussed below. To better demonstrate our implementation, next we describe the Linux congestion control architecture first. For the purpose of efficiency, the Linux implementation of congestion control is tightly coupled with other TCP features. We dedicate to keep compatibility with TCP and take advantage of certain features of TCP. As a result, it is neither reasonable nor effective to simply prune the congestion control from the Linux TCP implementation. Recall that we tend to implement VCP as a "layer 3.5" protocol, which means that once a packet arrives at the TCP layer after crossing the VCP layer, the information related to VCP is supposed to be lost.

Fortunately, as of 2.6.13, Linux supports pluggable congestion control algorithms enabling seamless integration of various congestion control algorithms with TCP stack. In this architecture, a variety of congestion control algorithms are organized via a data structure:

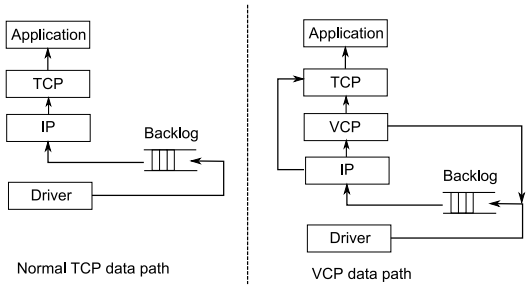

Fig. 1. Incoming VCP Data Packet Path

```
struct tcp_congestion_ops { ... }
```

By invoking the register function below, a group of TCP congestion control algorithms such as *reno*, *vegas*, and *bic* could be registered to TCP.

```
int tcp_register_congestion_control(
    struct tcp_congestion_ops *ca)
```

The latest registered one would be the active congestion control algorithm. Such architecture provides VCP with an elegant interface to interact with Linux's native congestion control scheme. Thus, we implement the congestion control of VCP as a pluggable congestion control algorithm of TCP which not only conforms to our principles mentioned earlier but is also very efficient.

More specifically, VCP consists of two components, working on *end-host* side and *router* side, respectively. Both components are implemented as LKMs, and hence can be transparently enabled and disabled.

C. VCP End Host Module

VCP defines a new IP packet type called VCP with the protocol number 200. While installing the VCP host module, by calling

```
int inet_add_protocol(
        struct net_protocol *prot,
        unsigned char protocol)
```

function, VCP is registered to IP and signals IP to deliver the packets with protocol number of 200 to the VCP's receiving handler. Meanwhile, the TCP's sending function is changed from *ip_queue_xmit()* to *vcp_queue_xmit()* in order to detour a TCP packet to VCP rather than IP. As noted earlier, VCP is implemented as one of the congestion control algorithms of TCP. Thus *tcp_vcp*, a new congestion control algorithm, is registered to TCP while installing the VCP module. *tcp_vcp* is responsible for updating the *cwnd* of TCP. In what follows,

we explain what happens to a TCP packet in terms of outgoing and incoming paths after enabling VCP .

- Outgoing packet path: While transmitting, a TCP packet generated by normal TCP stack is bypassed to VCP before it is forwarded to IP. In *vcp_queue_xmit()*, the member variable *sk_protocol* in *socket* data structure is changed from 6 (TCP) to 200 (VCP). Meanwhile, the ECN field is marked as "01". Then, the VCP packet is forwarded to IP for transmission by a direct function call *ip_queue_xmit()*. Thus, although VCP is embedded between TCP and IP as a "layer 3.5" protocol, there is no need for a new VCP packet header. Furthermore, there is nearly zero overhead introduced in the outgoing path of a TCP packet.
- Incoming packet path: While receiving, a VCP packet is delivered from IP to the VCP's receiving handler since VCP has been registered on top of the IP protocol. After saving the value of ECN bits, VCP forwards the packet to TCP, leaving the rest of processing of the packet to TCP. Fig. 1 illustrates the incoming data path of VCP.

D. VCP Router Module

As presented in [20], a VCP capable router should be able to (i) sample and compute network load on the network link, and (ii) intercept VCP packets and mark ECN bits.

Taking advantage of $qdisc$ and timers, sampling and computing network load could be easily achieved in Linux. With regard to handling VCP packets, $netfilter$ [18] can be used which is a built-in Linux kernel module allowing for intercepting and manipulating network packets. Via registering a $hook$ function at the point of "NF_IP_POST_ROUTING" on the router, VCP packets could be intercepted by $netfilter$ and forwarded to the VCP router module, wherein the ECN bits of the packet are marked as LOW_LOAD, HIGH_LOAD, or OVER_LOAD.

It is worth noting that such an implementation allows for the co-existence of VCP with standard transport protocols such as TCP and UDP. Importantly, the mixture of flows of different protocols (VCP, TCP, UDP) does not adversely affect the performance of VCP.

While computations of VCP parameters are involved with Floating-Point (FP) operations and although Linux kernel supports such operations, we do not use them in order to avoid the performance overhead associated with its use. Rather, we choose to transform all FP numbers to integers by simply amplifying all FP numbers by a factor of 1000. In fact, there are three places in VCP where FP computations are needed. One is for the calculation of load factor in the router and the other two are for the calculations of the MI/AI parameters. Since operands associated with these computations are in a small range, such transformation provides an acceptable precision.

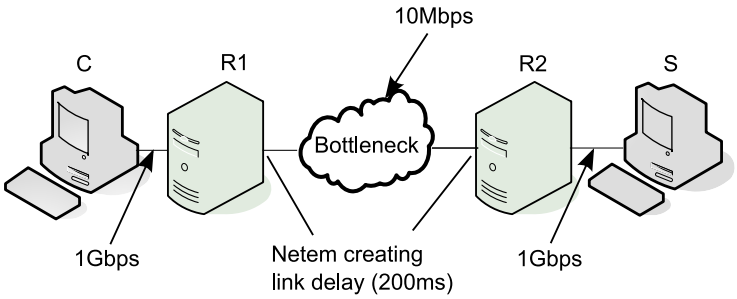

Fig. 2. The Experimental Environment

We note that there are several alternative approaches to implement VCP, i.e., (i) introducing a new TCP option similar to what is proposed in [27], or (ii) replacing the entire TCP protocol stack. While the former approach might require a separate implementation for each possible transport protocol, the latter approach introduces significant compatibility issues. In contrast, our approach is simple and efficient, while keeping compatibility with legacy TCP stacks.

IV. Experimental Studies

In this section, we present our experimental study conducted in a real testbed, comparing the performance of VCP, TCP Drop Tail (TCP/DT) and TCP Random Early Drop (TCP/RED). Fig. 2 shows our experimental setup. VCP end node code runs at end-hosts, i.e., the VCP Client (C) and the VCP Server (S). VCP router code runs on routers R1 and R2.

The bottleneck link is between R1 and R2, which is set to 10Mbps full-duplex via the $ethtool$ interface provided in most Linux distributions. The link delay is created via the $netem$ utility [7] on both routers to create 400ms Round Trip Time (RTT). An FTP server is set up at S using the $vsftpd$ built in the Fedora Core 5 distribution of Linux. Five FTP requests start from C to S simultaneously and the size of each requested file is 5MB. Both side links from C to R1 and from S to R2 are 1Gbps links operating in full-duplex mode.

To optimize the performance of TCP, a variety of TCP parameters are adjusted following the TCP performance tuning guides of [15], [26]. VCP parameters are set as presented in [20]. With regard to RED cases, the drop probability is set to 0.1, minimum and maximum queue size are set to one third and two thirds of the queue buffer size, respectively.

Specifically, our experiments analyze performance using three metrics:

- *Utilization:* The ratio of bandwidth consumed over the bottleneck link versus the bottleneck bandwidth, measured every RTT time unit. A ratio close to 1 is expected for VCP when it converges.

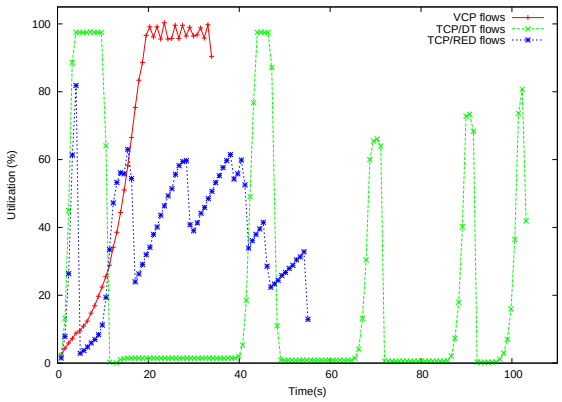

Fig. 3. Bandwidth Utilization of VCP, TCP/DT, TCP/RED

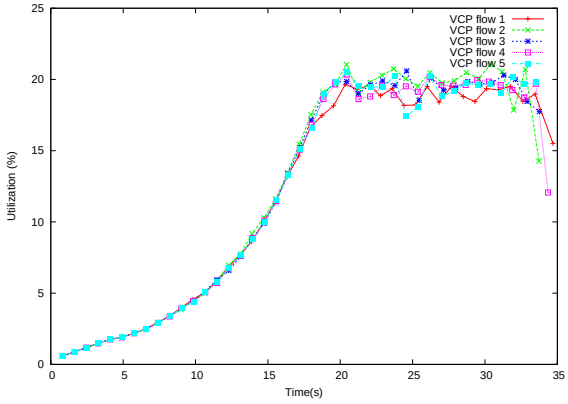

Fig. 4. Per-flow utilization (VCP)

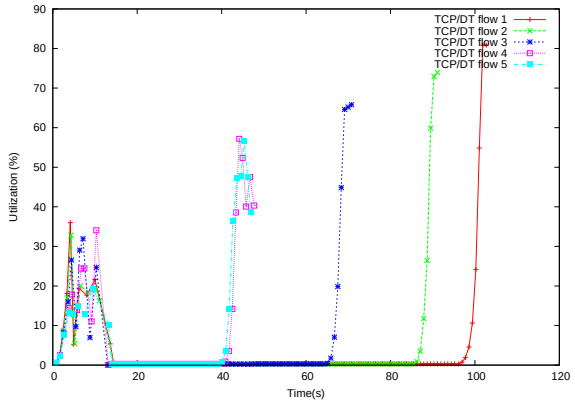

Fig. 5. Per-flow utilization (TCP/DT)

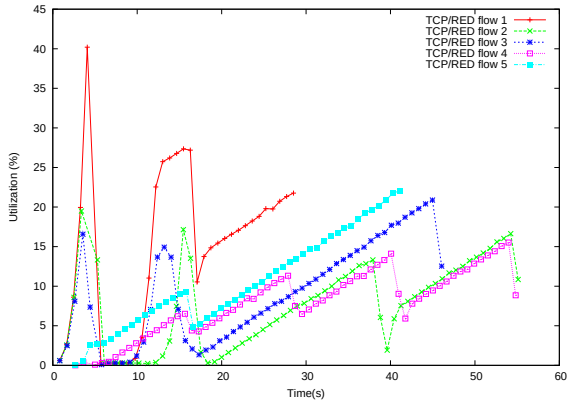

Fig. 6. Per-flow utilization (TCP/RED)

- *Per-flow utilization:* The ratio of each flow's bandwidth consumption versus the bottleneck bandwidth. VCP is supposed to demonstrate nearly the same ratio among all flows.
- *FTP elapsed time:* The average elapsed time for downloading 5 files via FTP. If VCP is more effective, shorter elapsed time is expected.

The experimental studies are two fold, one for a wired bottleneck link and another for a wireless bottleneck link. The wireless link effect is emulated via another LKM that implements the Gilbert-Elliott error model developed in our previous work [22] as a mean of capturing temporally correlated fading characteristics of wireless links. Using this emulated wireless network, we validate the simulation results in our earlier work of [23]. First, we measure the performance of VCP, TCP/DT, and TCP/RED in a wired environment to build our baseline and validate our implementation. Next, we explore the performance of VCP and TCP in wireless environments with a variety of parameter settings.

A. Performance in Wired Networks

We note that the purpose of this study is not to comprehensively evaluate the performance of VCP, but how VCP performs in wireless networks where non congestion loss is common. As a result, VCP might not demonstrate its full performance potential due to lack of fine-tuning. With proper fine-tuning, we believe that VCP is able to perform better than the results reported in this paper.

Fig. 3 shows a significant performance gap between TCP and VCP. As expected, VCP consistently achieves high bandwidth utilization and good fairness. In contrast, TCP/DT illustrates an unbalanced bandwidth allocation behavior between flows. Although TCP/RED significantly improves the performance of TCP/DT, it remains ill-behaved in terms of its fairness characteristic. Fig. 4, 5, and 6 illustrate per-flow utilizations of VCP and TCP. In the duration of downloading, all VCP flows consume bandwidth evenly, while TCP flows exhibit severe oscillations. In terms of completion time, the average time for VCP to download five files is 32.1 seconds while TCP/DT requires an average of 71 seconds and TCP/RED requires an average of 42.5 seconds. Furthermore, all VCP flows complete around the same time, while TCP flows finish with significant deviations. In the figure, the end point of each curve denotes the moment when all FTP flows finish. Fig. 7, 8, and 9 show the $cwnd$ measures of VCP and TCP, respectively.

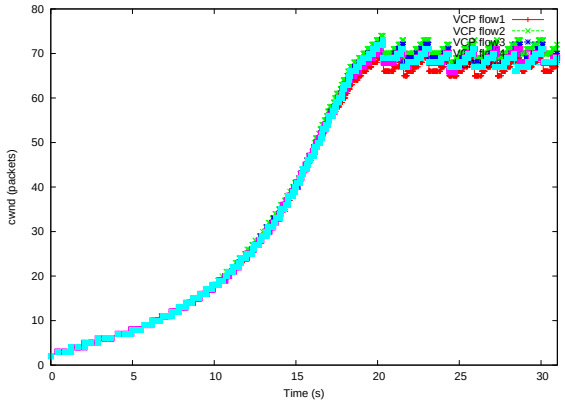

Fig. 7. Congestion windows size measures (VCP)

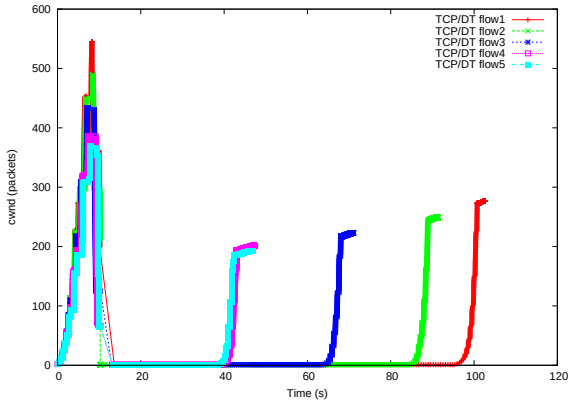

Fig. 8. Congestion windows size measures (TCP/DT)

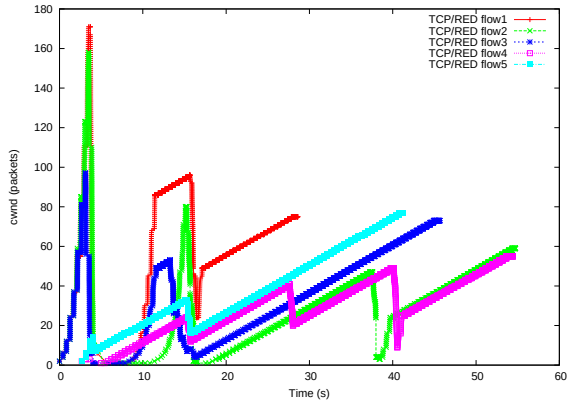

Fig. 9. Congestion windows size measures (TCP/RED)

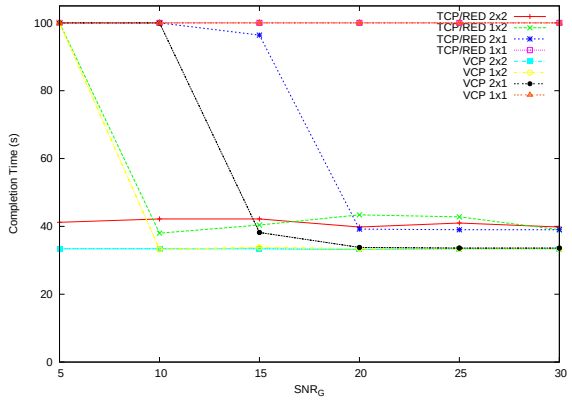

Fig. 10. FTP Completion Time of VCP and TCP/RED in Wireless Networks

B. *Experimental Evaluation with the Wireless Emulator*

This section compares the performance of VCP and TCP approaches in emulated wireless networks formed by Multiple-Input Multiple-Output (MIMO) links with/without FEC schemes applied at the link layer. We emulate the wireless link effect over the testbed noted earlier by applying an LKM module implementing the Gilbert-Elliott error-model over the bottleneck link.

In the following section, the configuration of error-model and the description of parameters follow those of [23].

1) The Effects of MIMO: Fig. 10 compares the completion time of VCP with TCP/RED at various antenna configurations. In all cases, VCP outperforms TCP/RED, achieving 20% faster transmission. As illustrated, the performance is directly related to the quality of the channel. For small values of SNR_G representing low channel quality, the value of Packet Error Rate (PER) is close to 100% and thus yields a very long completion time. As the quality of channel improves, the value of PER improves eventually approaching zero. For different antenna configurations, the performance of 1x1, 2x1, 1x2, and 2x2 MIMO links are in an ascending order as the result of improving SER from the former to the latter configuration. For proper scaling of the performance gap, completion times larger than 500 seconds are scaled down to 100 seconds.

We observe a very interesting phenomenon, that the completion time of TCP/RED is not proportional with SNR_G. As the link quality improves, the loss results from the congestion decreases. However, while the link quality drops for lower values of SNR_G, random bit errors introduce more packet loss somewhat limiting the explosion of the $cwnd$ and thus achieving a RED-like effect, i.e., by dropping packets earlier congestion losses are reduced. As a result, better completion times are achieved.

2) The Effects of FEC: As reported in [23], applying the same level of FEC protection to ACK packets as data packets results in significant ACK packet loss. The latter is due to the fact that short ACK packets are corrupted with a smaller number of bit errors. As verified by our experiments, this phenomenon can significantly affect the performance of VCP as it heavily relies on the congestion information in the ACK packets. Due to space limitation, we only report the results with different FEC rates applied to data and ACK packets.

Fig. 11 shows the effects of enabling link layer FEC for a 2x1 MIMO link. Note from the figures that introducing a small percentage of FEC at the link layer can significantly

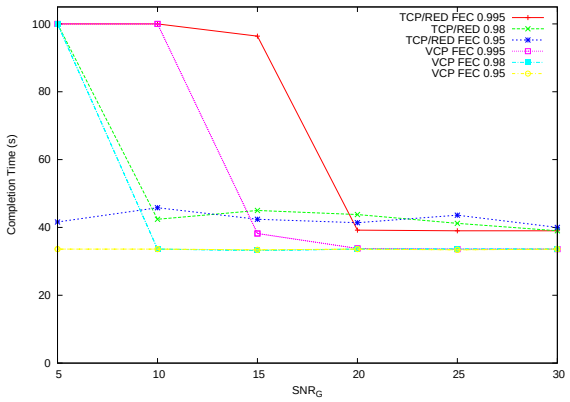

Fig. 11. FTP Completion Time of VCP and TCP/RED in Wireless Networks

improve the performance of VCP. While not shown due to the shortage of space, we have observed similar results for other configurations of MIMO links. Comparing different configurations of MIMO links, we observe once more that the performance of 1x1, 2x1, 1x2, and 2x2 are in an ascending order.

V. Conclusion

In this paper, we reported the results of our implementation of VCP and an experimental study on its performance in a real network testbed. Our implementation was transparent to applications, compatible with legacy TCP stack, and allowed for the co-existence of VCP with standard transport protocols such as TCP and UDP. We conducted our experimental study in a wired testbed consisting of Linux nodes realistically emulating the fading characteristics of wireless links in the Linux kernel.

Our experiments' results demonstrated that (i) the need for protecting protocol's metadata as well as data against bit errors, and (ii) that VCP represents a high performing yet practical congestion control protocol for encrypted wireless networks.

Our future work includes optimizing the performance of VCP by parameter fine-tuning, as well as a comprehensive performance evaluation versus XCP and other TCP/AQM mechanisms.

References

[1] S. Athuraliya, V. H. Li, S. H. Low, and Q. Yin. REM: Active queue management. *IEEE Network*, 15(3):48 – 53, May/June 2001.

[2] S. Bhandarkar, S. Jain, and A. Reddy. Improving tcp performance in high bandwidth high rtt links using layered congestion control. In *Proc. of the PFLDNet'05*, Feb. 2005.

[3] D. Chiu and R. Jain. Analysis of the increase/decrease algorithms for congestion avoidance in computer networks. *J. of Computer Networks and ISDN*, 17(1):1 – 14, June 1989.

[4] S. Floyd. HighSpeed TCP for large congestion windows. Aug. 2002.

[5] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(9):397–413, Aug 1993.

[6] M. Goutelle, Y. Gu, and E. He. A survey of transport protocols other than standard tcp. In *Data Transport Research Group*, 2004. work in progress, April 2004. https://forge.gridforum.org/forum/forum.php?forum id=410.

[7] S. Hemminger. Network emulation with netem. In *Proc. LCA, 2005*, Apr. 2005.

[8] V. Jacobson. Congestion avoidance and control. In *ACM SIGCOMM '88*, pages 314–329, Stanford, CA, Aug. 1988.

[9] A. Jain and S. Floyd. Quick-start for tcp and ip. In *IETF Internet Draft draft-amit-quick-start-02.txt*, Oct. 2002.

[10] C. Jin, D. Wei, and S. Low. Fast tcp: Motivation, architecture, algorithms, performance. In *Proc. of the Infocom 04*, 2004.

[11] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *Proc. ACM SIGCOMM, 2002*, Aug. 2002.

[12] T. Kelly. Scalable tcp: Improving performance in highspeed wide area networks. Feb. 2003. Available at http://wwwlce.eng.cam.ac.uk/ctk21/scalable/.

[13] S. Kunniyur and R. Srikant. Analysis and design of an adaptive virtual queue (avq) algorithm for active queue management. In *Proc. of the 2001 ACM SIGCOMM Conference*, 2001.

[14] D. Leith and R. Shorten. H-tcp: Tcp for high-speed and long-distance networks. In *Proc. of the PFLDNet'04*, Feb. 2004.

[15] J. Mahdavi. Enabling high performance data transfers on hosts. *technical note, Pittsburgh Supercomputing Center*, Dec. 1997. Available at http://www.psc.edu/networking/perf_tune.html.

[16] K. Ramakrishnan, S. Floyd, and D. Black. The addition of explicit congestion notification (ecn) to ip. In *IETF RFC 3168*, 2001.

[17] I. Rhee and L. Xu. Cubic: A new tcp-friendly high-speed tcp variant. In *Proc. of the PFLDNet'05*, Feb. 2005.

[18] R. Russell and H. Welte. Linux netfilter hacking howto. July 2002. Available at http://www.netfilter.org/documentation/HOWTO//netfilter-hacking-HOWTO.html.

[19] B. Wydrowski and M. Zukerman. Maxnet: A congestion control architecture for maxmin fairness. *IEEE Comm. Letters*, 6(11):512 – 514, Nov. 2002.

[20] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman. One more bit is enough. In *Proc. ACM SIGCOMM, 2005*, Aug. 2005.

[21] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control (bic) for fast long-distance networks. In *Proc. of the Infocom 04*, 2004.

[22] H. Yousefi'zadeh and A. Habibi. A performance comparison study of end-to-end congestion control protocols over mimo fading channels. In *Proc. of the IEEE MILCOM, 2006*, Oct. 2006.

[23] H. Yousefi'zadeh, X. Li, and A. Habibi. An end-to-end cross-layer profiling study of congestion control in high bdp wireless networks. In *Proc. of the IEEE WCNC, 2007*, Mar. 2007.

[24] -. UCB/LBNL/VINT Network Simulator - ns (version 2). Available at www.mash.cs.berkeley.edu/ns/.

[25] -. Global information grid net-centric implementation document: Quality-of-service (t300). *FOUO Pre-Coordination Plan Document*, Aug. 2005. https://disronline.disa.mil/a/DISR/docs/T300_V1-0_FINAL.pdf.

[26] -. Tcp tuning guide. Nov. 2005. Distributed Systems Department, Available at http://dsd.lbl.gov/TCP-tuning/TCP-tuning.html.

[27] Y. Zhang and T. Henderson. An implementation and experimental study of the explicit control protocol (xcp). In *Proc. IEEE INFOCOM, 2005*, Mar. 2005.