Fairness Attacks in the Explicit Control Protocol

Christo Wilson, Chris Coakley and Ben Y. Zhao
Computer Science Department, U. C. Santa Barbara
{*bowlin, ccoakley, ravenben*}@*cs.ucsb.edu*

***Abstract*— Protocols such as the Explicit Control Protocol (XCP) use explicit router feedback to guide endpoint transmission rates for near-optimal capacity utilization and fairness. However, non-cooperative end hosts can manipulate and ignore feedback to either obtain unfair advantages over cooperative hosts, or perform denial-of-service attacks on intervening network links. In this paper we explore the methodology behind, and construct working examples of different attack vectors on XCP, including both cheating senders and receivers. Through detailed simulations in ns, we show that misbehaving users can dominate bandwidth allocation on shared links, and our strategies allow them to successfully allocate bandwidth by either sharing or selfishly competing for the bottleneck bandwidth capacity.**

I. Introduction

A number of recent projects proposed effective transmission control protocols to manage wide-area network congestion. Both simulations and measurements have confirmed the effectiveness of protocols that leverage explicit congestion feedback from routers, including ECN and Explicit Control Protocol (XCP) [1]. For example, XCP decouples fairness and efficiency control, resulting in asymptotic optimal bandwidth utilization and responsive adaptation to network congestion.

XCP packets include a congestion control header that includes the sender's current throughput, round trip time (RTT) estimate, and a reverse feedback value from the network intended for the packet's recipient. While in transit, XCP-enabled routers can modify the throughput value in the congestion header to reflect their desire for the sender to increase or decrease its congestion window size due to locally observed network conditions. On receipt of a packet, the destination host uses the reverse feedback value to update its own congestion window, then copies the modified throughout value into the reverse feedback field of the next packet to be sent. Feedback for each host is built up as they send packets, and the recipient forwards network feedback back to the sender during acknowledgment through the reverse feedback header field. This feedback system enables routers to control exactly how much data flows through them while storing no per-flow state. Routers can prevent congestion by instructing hosts to reduce their windows and can keep utilization high in the event of unused bandwidth by instructing hosts to increase their windows. Fairness is imposed by the same mechanism; any host which reports an anomalous throughput value will be told to equalize their window size with all other hosts that are sharing the same link.

While XCP works exceptionally well when all peers follow the recommendations of network feedback [1], it creates an environment that can easily be exploited by a small number of misbehaving hosts. Endpoint peers can ignore or overwrite router feedback values for two possible goals. In the common scenario, one or both endpoints can manipulate network feedback to boost their own share of the network bandwidth. In addition, a connection receiver can mislead data senders to generate a denial-of-service attack on other hosts or routers on the end-to-end path. Given the ubiquitous high-bandwidth demands of peer-to-peer content distribution programs, together with the prevalence of Internet DoS attacks, we believe any XCP deployment will be quickly exploited, resulting in unfair bandwidth allocation and difficult to detect DoS attacks.

In this paper, we seek a better understanding of different classes of attacks against the XCP protocol. We explore the methodology behind, and construct working examples of different attack vectors on XCP, including both cheating senders and receivers. We explore aggressive bandwidth stealing attacks that allow cheating attackers to quickly usurp available bandwidth from well-behaved XCP peers, and how cheaters can "cooperate" or compete amongst themselves for bandwidth while minimizing network congestion from their aggressive tactics. Through simulations using ns-2, we show that a single misbehaving user can dominate bandwidth allocation in a large group of flows. Additionally, our strategies allow attackers to successfully allocate bandwidth by either sharing or selfishly competing for the bottleneck bandwidth capacity.

The remainder of the paper is structured as follows. We describe related work in Section II, followed by Section III, where we describe our strategies for exploiting XCP through non-cooperation. In Section IV, we evaluate our strategies through both simulation. Finally, we discuss implications of our results and conclude in Section V.

II. Related Work

The designers of XCP were aware of the possibility of malicious XCP hosts. Section 7 of the XCP draft [2] proposes using monitoring techniques in edge routers to police the network and provide security. Routers locate unresponsive nodes by monitoring their compliance with specific feedback messages. Hosts which fail to respond could then be throttled or denied access as seen fit. Since this technique also relies on throughput and round trip time estimates reported by clients to gauge responsiveness to feedback, it is also vulnerable to misinformation from clients.

There are a number of TCP-based attacks that are related to our work, the most pertinent of which are receiver-side, ACK-spoofing vulnerabilities, first uncovered in [3]. By sending flurries of spurious ACK packets, or optimistically ACKing unreceived data, a TCP receiver can cause the sender to inflate their congestion window prematurely, thus speeding up the transfer of data [3]. This attack works because the receiver is the purveyor of the feedback (the ACK packets) which dictate the growth of the sender's congestion window. Protection from misbehaving receivers can be implemented through a cumulative nonce, where the receiver must prove in-order reception of previous packets by providing the sender with a cumulative XOR value of random numbers embedded inside data segments (nonces). [4] explores the potential consequences of widespread TCP receiver misbehavior.

Endpoint misbehavior attacks are also relevant to other congestion control protocols. The attack vectors we discuss are relevant to all protocols that rely on user cooperation, including traditional Explicit Congestion Notification (ECN) [5], [6] and XCP variants such as VCP [7]. Others have begun to address similar issues in related protocols by proposing tampering resistant variants such as Robust TCP-Friendly Congestion Control (RTFRC) [8].

A considerable amount of work has focused on different aspects of the XCP protocol. Low et al. present a fluid dynamic model of representing abstract XCP traffic patterns [9], while others have analyzed XCP's properties in the presence of capacity estimation errors [10]. Zhang and Henderson created a test implementation of XCP on top of the Linux kernel and performed test bed analysis of the protocol to verify the simulation results [11]. While initial tests confirmed the results from simulation, the authors also point out XCP's unpredictable characteristics over lossy media and in the presence of non-XCP enabled router queues. The authors show that XCP routers are extremely sensitive to the end-host feedback, and recognize the potential for DoS attacks from malicious XCP hosts.

Although our study does not include measurements on the impact of XCP-based DoS attacks, prior studies have mentioned the potential impact of these attacks [11]. Existing work describe egregious examples of denial of service attacks in general [12], while others present a taxonomy of denial of service attacks and countermeasures [13].

III. Design of Attacking Hosts

A number of protocols rely on router feedback to guide congestion control, including Explicit Control Protocol (XCP) [1], Variable-structure congestion Control Protocol (VCP) [7], and Explicit Congestion Notification proposals [6]. While these protocols avoid reliance on dropped packets as congestion feedback, they all rely on the compliance of endhosts to adjust their send rates to alleviate congestion. Unfortunately, in the presence of malicious users, the fast convergence of XCP works to the disadvantage of normal users by quickly dropping their send rates.

In this section, we describe design considerations for attack strategies on XCP, and use them to construct both sender and receiver-based attacks. We describe the attacks in detail, and further evaluate them via ns-2 in Section IV.

A. Bandwidth Contention Amongst Cheaters

The singular goal of a cheating XCP host is to secure as much bandwidth for itself as is possible. There are two reasons to do this:

- To speed up data transfers
- To cause a denial of service attack

In order to assume control of bandwidth the malicious host must increase the size of its congestion window so that it is sending as much data as possible. This can also be done by proxy, *i.e.* by deceiving a sender into performing the same increase of their congestion window at the receiver side attacker's behest. An example of this type of attack is Optimistic ACKing [3].

A single, cheating XCP host is able to flood as much data onto the network as there is bandwidth at the most constrained link on its end-to-end path. This is possible due to XCP's feedback mechanisms: as the cheater assumes control of more and more bandwidth all other well behaved flows will be instructed to back-off their send rates lest they cause congestion. Thus, XCP enabled networks present an ideal place to cheat since well behaved hosts are guaranteed to relinquish bandwidth gracefully.

The difficulty in creating a robust cheater lays in the problem it itself creates: how do cheaters compete amongst themselves for limited resources? If one cheating host already has control of the bandwidth on a given network, the arrival of another cheater attempting to increase its window size will surely result in congestion, since both nodes are aggressively competing for bandwidth. The end result is packet drops, which are undesirable for legitimate hosts and cheaters alike.

One approach is for attackers to accurately gauge instantaneous end-to-end bandwidth conditions by leveraging dedicated capacity estimation tools such as CapProbe [14] or utilize a passive, in-band capacity estimation protocol like the Probe Control Protocol [15]. Being able to form capacity estimates allows an attacker to assume control of bandwidth without inciting congestion. Unfortunately, a known limitation of capacity estimation methods is that they are unable to provide accurate appraisals on links that are heavily loaded by unresponsive flows, which are the exact conditions that occur when another attacker is present on the network path.

A second approach is to simply copy TCP's congestion control algorithm. Although this algorithm does incur some overhead in the form of dropped packets, it is obviously still quite effective at allowing a host to ramp up window size in a controlled manner. If all attackers on a given network link implement this algorithm then the system approximates the conditions on a normal, TCP dominated network link: each attacker repeatedly competes for bandwidth and backs off when congestion is detected. Our simulation results demonstrate that

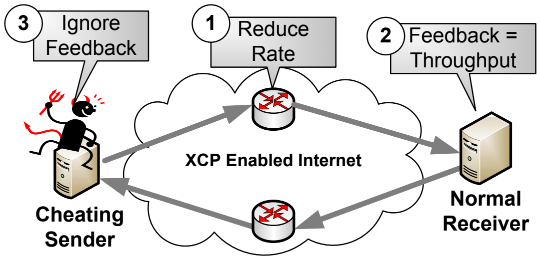

Fig. 1. A malicious sender-based attack.

attackers using this algorithm are effective at starving the bandwidth of any normal XCP flows sharing the bottleneck link.

There is still a lingering problem with emulating TCP congestion control in the context of malicious XCP clients. Using this method assumes that all attackers are still responsive to congestion, even if they are unresponsive to XCP feedback. However, making any assumptions about the complacent behavior of an attacker is a fallacy. If any attacker on a link is totally unresponsive then no amount of probing by any means will be able to secure additional bandwidth for an emergent cheater. Conversely, additional probing simply causes packet drops and lower overall throughput for the congestion controlled attacker. Other than resorting to a network flooding war of attrition against the unresponsive flow in an attempt to force a response from it, a congestion controlled attacker can not win against a fully unresponsive one.

This realization leads to the development of a third attacking model: the malicious host probes for bandwidth once, then assumes control of whatever it can find and ignores network conditions until it is done transferring data. This final approach allocates bandwidth on a first-come-first-serve basis, where the attacker who is first to probe across a bottleneck link is sure to take control of the majority of the bandwidth present; all subsequent attackers will be forced to split the remaining bandwidth. The one instance where this method of attack is sub-optimal is when an attacking flow in control of a large portion of the available bandwidth stops sending. When this happens, if there are other attackers present on the link, it is in their best interest to take control of the resources which were just freed, but since they are not actively probing they may not be capable of doing this. This issue can be resolved through passive monitoring: attackers which notice a sudden decrease in RTT values or surge in positive XCP feedback can surmise that new resources have become available and re-enter the initial bandwidth probing phase.

B. Sender-side Attacks: the Abusing Host

With the aforementioned principals in mind we construct an abusing XCP client which (mostly) ignores XCP feedback and attempts to cheat by sending as much data as possible. The underlying methodology for our implementation is the standard TCP congestion control scheme. The attacker always applies positive XCP feedback if it is received, which effectively allows the attacker to skip the TCP slow-start phase. Once the XCP feedback reaches zero or is negative it is ignored and the standard TCP additive increase algorithm is used to continue growing the attackers congestion window size. The additive congestion window growth equation is:

$$new\_cwnd = old\_cwnd + 1/old\_cwnd$$

This continues until a packet is retransmitted, indicating that there has been a loss due to congestion. At this point the congestion window size is reset to 1, which is standard TCP procedure when a retransmission timer expires. The congestion window size remains as 1 for one RTT, which has the effect of allowing any overflowing network queues to recover. After this time period the congestion window size is set to 50% of its value prior to backoff, in accordance to standard TCP multiplicative decrease procedures. It is worth noting that a short timer which is triggered on packet retransmission is used to prevent several sequential retransmissions from triggering multiple 50% backoffs.

In our testing we evaluate two different types of abusing hosts: one that continuously probes and one that locks its congestion window size after the initial probing. In the former build, the attacker resumes using the additive increase equation to probe for bandwidth after the congestion window has been reset to 50% due to a packet loss. In the latter build, the attacker does not resume additive increase; instead it simply locks the size of its congestion window at 50% and ignores all feedback. While a locked attacker is in this state, it passively monitors the average RTT of its packets in order to guage whether additional network resources are available. If a locked attacker observes a reduction in RTT by at least 25% over a period of less than a second it will unlock its congestion window and resume probing using the additive increase equation.

This method of passively monitoring for additional bandwidth works since we assume that each attacker is utilizing a network link who's bandwidth is constrained by one of two things:

- Maximum network bottleneck bandwidth
- The presence of other attackers

In the first case, constraint by physical network characteristics means that the attacker is the first and only cheater present. In both cases all available bandwidth will be used, either by the attacker in question or by a combination of attackers. Although an individual attacker can not determine which of the these two conditions it is currently operating under we know that all bandwidth will be used, therefore a dramatic reduction in RTT can only be indicative of the freeing of network resources. In this case it is essential to begin probing again so the resources can be appropriated.

C. Receiver-side Attacks: the Lying Host

The previous section describes the construction of an abusive XCP client which ignores feedback in order to send data

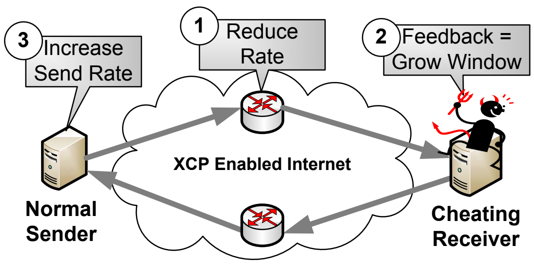

Fig. 2. A malicious receiver-based attack.

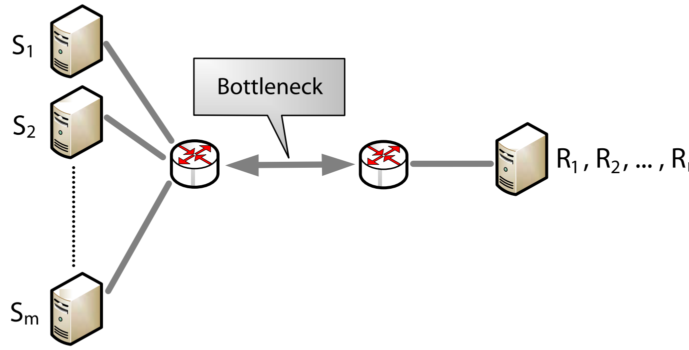

Fig. 3. Network topology used during ns-2 simulations. Multiple senders direct traffic across a bottleneck link to traffic sinks on the opposite side.

at arbitrary rates. In the converse situation, it is possible to construct a lying XCP host which manipulates XCP feedback in such a way as to cause data to be sent to itself at arbitrary rates. Implementing this attack is fundamentally different than its sender-side counterpart since the attacker does not have direct control over the sending flows congestion window size. Instead, the liar must deceive the benign sender by reporting altered reverse feedback values to it. Since all XCP feedback that is destined for the sender must first arrive at and be forwarded by the receiver, this process is relatively straightforward.

The calculation performed by XCP to determine the new congestion window size based on incoming feedback is as follows:

$$new\_cwnd = old\_cwnd + feedback \times rtt/packet\_size$$

The cheating receiver can predict what the sender's congestion window size will be after receiving any given piece of feedback by using this same equation, substituting in its previous congestion window estimate as old_cwnd and its own round trip time estimate as rtt. On the creation of a new flow old_cwnd is assumed to be 1 initially. Using this method the receiver can calculate whether the feedback it is going to forward will cause the sender to increase their congestion window size, in which case the value is sent unaltered, or whether the sender's window growth has halted. In the latter case, the same principal used by the abusing attacker implementation is used: substitute the reverse feedback for a new value will which cause the sender to additively increase the size of their congestion window. On incoming packet drops the lying receiver halves their estimate of the sender's congestion window to remain consistent with what the sender will do locally. Internally, the attacker also keeps track of the desired congestion window size for the sender, which also gets halved in the event of a packet drop. A short timer is used to ensure that sequential drops do not trigger multiple reductions.

Just as with the abusive attacker, we created versions of the lying attacker which continuously cause the sender to probe for additional bandwidth, and a version where the congestion window size is locked after the first packet drop.

IV. Simulation Results

A. Simulation Environment

For our tests, we modified ns-2.30 [16] to enable explicit control of the congestion window, delta throughput, rtt, and reverse feedback values. We added callback methods so that end hosts could set these values both internally and in the appropriate packet locations before and after the XCP code processed them. This was done for both traffic generators and sinks. These changes allowed all attack techniques to be implemented in the same TCL scripts that controlled the simulation. Our TCL scripts implementing the attackers and running the tests, as well as patches to ns-2 to enable the necessary additional functionality are available for download[1].

Each simulation was performed with a 20Mb network bottleneck, 10ms latency, 1K packet size, using the droptail queuing method. The network topology was derived from [1] and is shown in figure 3.

The version of XCP implemented in ns-2 is Dina Katabi's own reference implementation. As written, the XCP layer sits directly above TCP and defers all congestion control to it. [2] leaves the question of how to integrate XCP into the network protocol stack ambiguous; the existing Linux implementation [11] operates as a parallel architecture to TCP and does not rely on its congestion control mechanisms at all. Although our attackers works in concept regardless of underlying structure, the implementation is tied to the specific context of the ns-2 implementation and would need to be modified in order to function correctly should the layer beneath XCP be changed.

B. Abusing Host Simulation

We evaluate the abusive attacker with congestion window locking behavior in several different contexts. The first is when there is only a single attacker with no other flows present, and the second is when there are a varying number of non-cheaters. Figures 4 and 5 demonstrate the effectiveness of this

[1]Code available at `http://current.cs.ucsb.edu/downloads.html`

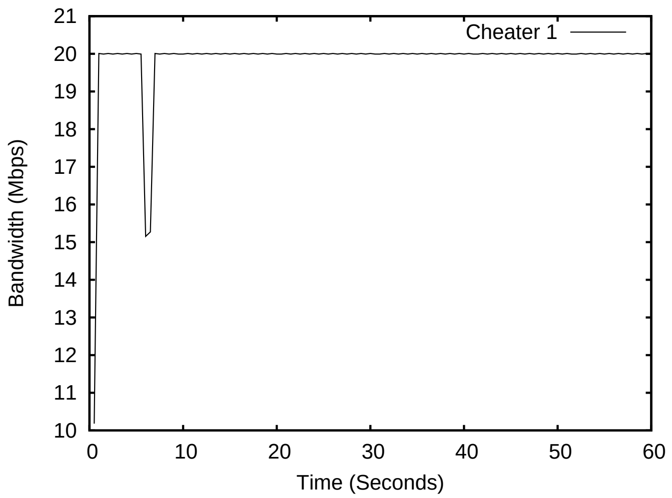

Fig. 4. A solo abusive attacker. The single drop in bandwidth usage is due to the attackers one attempt at probing for additional bandwidth, which triggers packet losses due to the bandwidth ceiling at the bottleneck link.

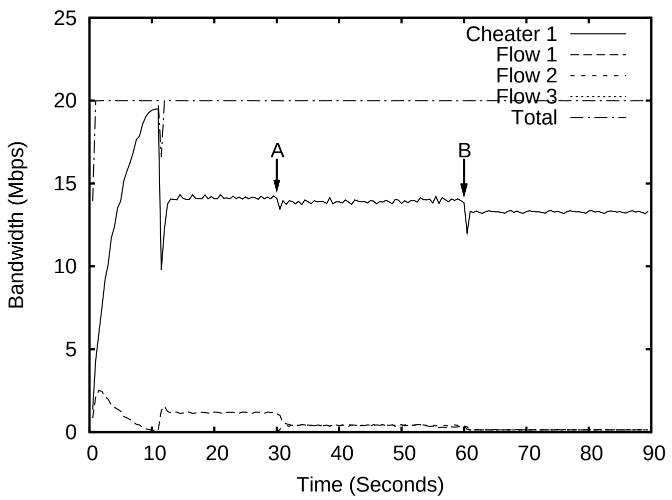

Fig. 5. A solo abusive attacker with 49 total normal hosts. Initially there are at 5 flows present: at 30 seconds (label A) 10 additional flows join, and at 60 seconds (label B) 35 more flows join. Each graphed Flow is from a different wave.

attack. Note that in Figure 4 the attacker fails to achieve the consistently perfect utilization of the network that a normal single XCP flow would achieve only during the initial probe period. After that the congestion window size is locked, which halts probing behavior, and positive XCP feedback informs the attacker of the optimal congestion window size to assume. Figure 5 demonstrates the effectiveness of a cheater with 4, 14, and 49 normal XCP hosts on the network, with the changes in density occurring every 30 seconds. At no point does the cheating host fall below the bandwidth that would be assigned fairly. Network utilization is consistently high, while packet drops for the attacker are almost non-existent at 2 for the duration of the simulation.

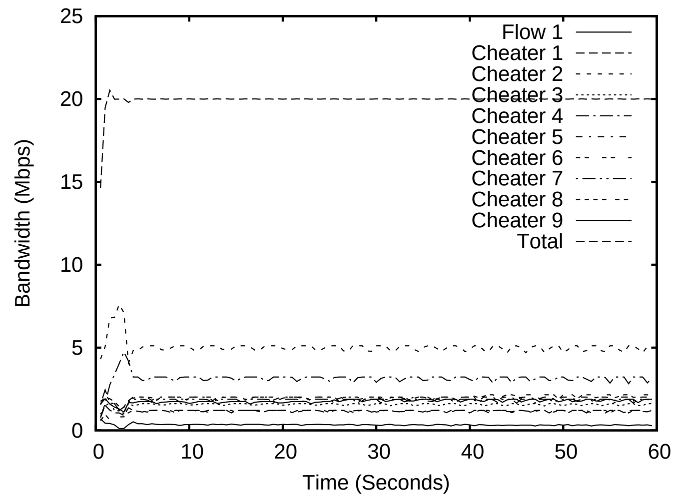

Fig. 6. Nine abusive attacker with locking behavior and one normal flow. The normal flow is not allotted any bandwidth and is therefore the bottommost line on the graph.

Figures 6 and 7 demonstrate the effectiveness of the sender-side abusive attack in the presence of other cheaters for both our continuously probing and window locking implementations. Although overall network utilization remains high and the one normal XCP flow present in the simulation is successfully stifled in both simulations, the constantly probing attackers do not show any stability. The chaotic, TCP like, nature of this system leads to considerable congestion variance and packet drops, as can be seen in Figure 8. In contrast the abusers with locking behavior act as expected: one attacker manages to grab one quarter of the available bandwidth by being the first to move (by milliseconds), and the other attackers are left to apportion the remaining bandwidth amongst themselves. Ten normal XCP flows sharing a 20mbit bottleneck would all settle at using 2mbits apiece, which is approximately what the majority of the attackers in this simulation settle at, demonstrating that abusive behavior is at least as effective as normal XCP behavior even in the presence of other attackers.

Comparing these two sets of simulation results reinforces our initial assumption that attackers with locking behavior would demonstrate superior performance when compared to constantly probing abusers. The latter implementation incurs severe packet drops, which is antithetical to the benefits of using XCP in the first place. In contrast, the window locking abusive attackers never drop more than two packets per flow.

One anomaly worth discussing is the fact that network utilization climbs above 100%. This is due to the fact that utilization is measured as total outgoing bandwidth. At points above 100%, the router queues are being filled up. This event is always followed by one or more packet drops, which is followed by one or more flows backing off.

Figure 9 demonstrates the effect of staggering the entrance of abusive attackers with window locking behavior into the

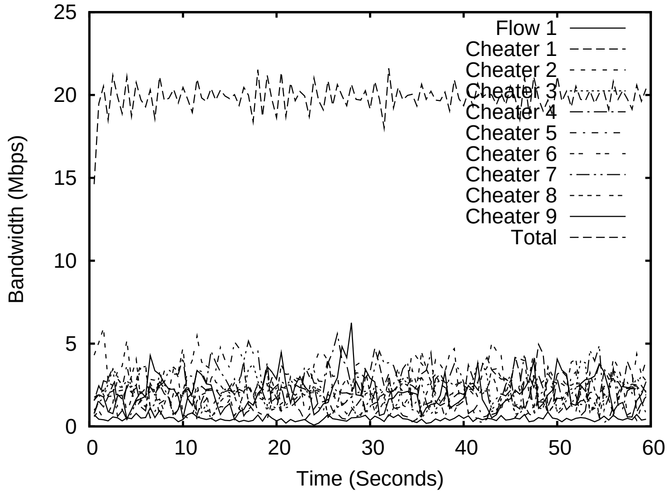

Fig. 7. Nine abusive attackers without locking behavior and one normal flow. Constant probing for bandwidth by each attacker results in chaotic backoff behavior, although overall utilization remains high.

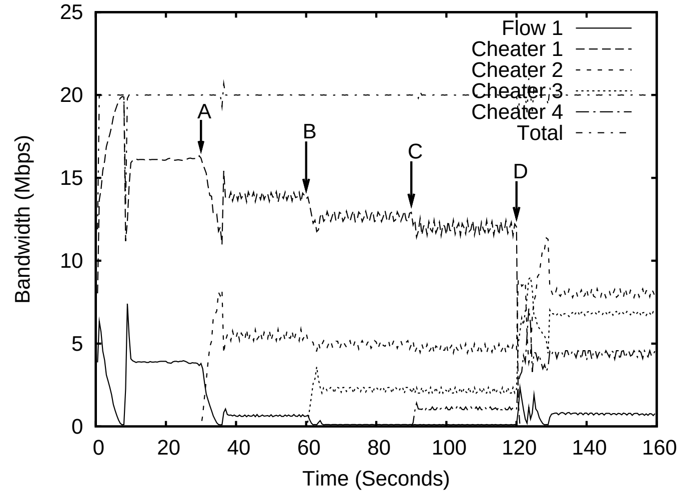

Fig. 9. Four abusive attackers and one normal flow. Initially only one attacker is present; another joins every 30 seconds (labels A, B, and C). The most successful abusive flow (Cheater 1) stops sending at 120 seconds (label D), causing the other abusers to restart probing behavior and seize additional resources which were previously unavailable.

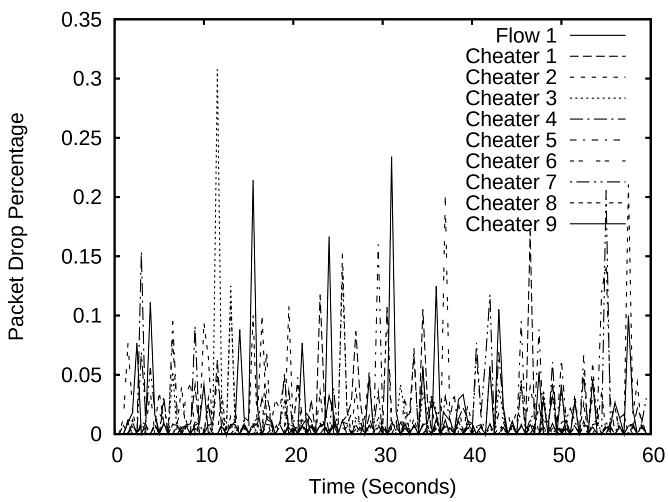

Fig. 8. Average packet drops for nine abusive attackers without locking behavior and one normal flow.

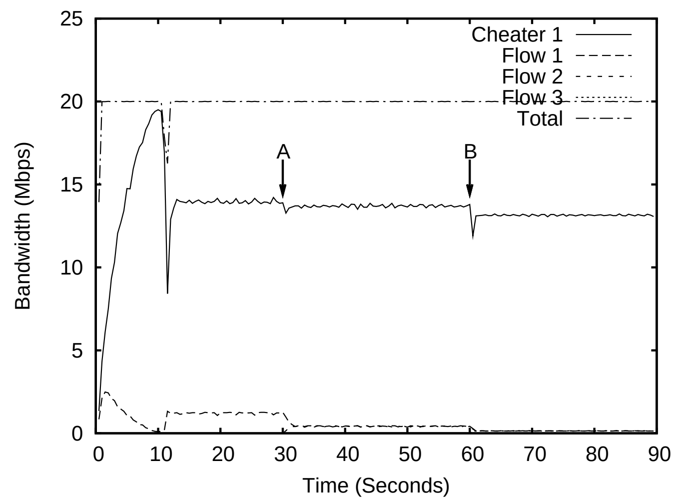

Fig. 10. One lying attacker with 49 total normal hosts. Initially there are at 5 flows present: at 30 seconds (label A) 10 additional flows join, and at 60 seconds (label B) 35 more flows join. Each graphed Flow is from a different wave.

simulation, as well as the consequences of an attacking flow halting. As expected, the attackers divide up the available bandwidth on a first-come-first-served basis, while managing to almost totally quash the one normal flow present. At 120 seconds Cheater 1, the most successful attacker, stops sending, freeing up a substantial amount of bandwidth. Immediately, the other cheaters observe a drop in average RTT and recommence probing. No singular attacker manages to obtain control over the open bandwidth, although the three remaining attackers also do not divide it up evenly. Even when probing recommences at 120 seconds, no flow present in the simulation ever drops more than 2 packets during each period of additive increase.

C. Lying Host Simulation

Figure 10 demonstrates the effectiveness of the lying attack for receivers. Similar to figure 5, the network starts out with 1 cheater and 4 normal hosts. Normal hosts are added to the network in groups: 10 at 30 seconds and 35 at 60 seconds. The attacker maintains a hold on nearly all of the bandwidth no matter how many other flows are present. Packet drops for the length of the simulation are essentially nonexistent. Although not reproduced here, graphs of the lying attackers predicted congestion window size plotted alongside the manipulated senders actual window size reveal an almost perfect 1:1

mapping, meaning that the attackers strategy and predictive abilities are sound.

V. Discussion and Conclusions

A. Attacker Evaluation

The simulation results of our attackers exemplify their success at accomplishing the stated goal of their design: to control as much available bandwidth as possible and stifle legitimate XCP flows. Normal, responsive XCP traffic's compliance with feedback makes it an ideal target for exploitation since flows will gracefully scale back when faced with an unresponsive attacker. This enables malicious hosts to dominate network resources no matter how many other flows are present while incurring almost no penalties in the form of dropped packets or untenable latencies.

Since attacking the XCP protocol yields such high returns, there is significant incentive for the behavior to take place. Given that attackers can completely sap available network resources this inevitably leads to a situation where the only effective way to aquire bandwidth at all is to attack as well. This creates a negative feedback cycle which reinforces bad behavior as the only tenable form of action. Whether attackers work in concert together or not, *i.e.* their varying responses to congestion feedback, they still exhibit low overhead, high overall utilization, and total effectiveness at starving normal XCP flows. Our results show that even in the presence of window locking, fully unresponsive attacking flows, the other attackers still manage to gain more bandwidth than any normal flows which are present. Strikingly, as noted above, in this situation the majority of attackers end up securing about as much bandwidth as they would have in the case that there were no attackers at all and each host responded to XCP feedback appropriately. This simply reinforces the point that ironically, in the presence of even one attacker, the only way to achieve a "fair" portion of network throughput is to attack as well.

What is particularly alarming about our results is the effectiveness of the lying attacker. Just as with TCP optimistic ACK attacks, a lying XCP attacker can force a fully standards compliant XCP host to behave in arbitrarily non-compliant ways. This type of attack bolsters the ability of malicious users to leech off of content providers, appropriating all available bandwidth for their own selective data transmissions. It is also plausible to imagine a lying XCP receiver being used to initiate a reflector style denial of service attack, much like the smurf attack [17], by forcing several servers with large bandwidth capacity to direct excessively large amounts of data traffic at a specific source. Coupled with IP address spoofing this attack could be directed at any end-host, but even without it an attacker could cause a localized service outage for a website or other data provider by causing their servers to flood their own outgoing links to the Internet.

B. Mitigation Strategies

There are multiple potential strategies to detect attacking XCP hosts. One approach is to place monitors at every end point of the network [2], [1], [18]. [2] recommends a probabilistic approach to monitoring in an attempt to provide a scalable solution, though no analysis is provided. While we agree with the recommendation for monitors, we discuss the monitoring requirements for detecting both abuser- and lying-attackers and requirements for both local and remote detection. The rationale for remote detection is simple: we assume that it is possible for an attacker to use a compromised or unmonitored access point to the network.

Remote and local detection of an abusive attacker has the same requirements. The receiver, presumably a non-cheater, copies delta throughput into reverse feedback. The reverse feedback value is untouched by the network. The monitor (again, remote or local), must track the bandwidth of the flow and ensure that the reverse feedback values are being heeded. The remote monitor has a more difficult job, as an estimated half of an RTT will pass before a value is acted on. It is insufficient to merely record the reverse feedback being sent to a host for comparison against the cwnd value it will report in the XCP header of its next packet, since it would be trivial for an abusive attacker to augment their abilities to lie about their actual values as well. The only solution is rate monitoring, which unfortunately necessitates stateful overhead in all XCP enabled leaf routers.

Local detection of a lying attacker is easy. Since there are no XCP routers in between the monitor and the attacker, the incoming delta throughput must match the outgoing reverse feedback. If not, the monitored host is an attacker. Local monitoring is not a sufficient countermeasure against lying attackers: the bandwidth and denial-of-service effects impact the sender, who also typically incurs any bandwidth costs. Because the impacts are on the side of the sender, best practices dictate that monitoring and detection occur remotely (near the non-cheating sender). Unfortunately, detection is difficult for a number of reasons:

1) XCP routers alter the delta throughput value after it has passed the remote monitor in route to the receiver.
2) Packet drops, which only occur significantly on XCP in the presence of attackers, only indicate the presence of an attacker along the flow's path, not that a given receiver is an attacker.
3) Asymmetric traffic (small ACK packets vs. large data packets) makes modeling the remote attacker difficult, even with symmetric paths. For this reason it is not sufficient for the monitor near the sender to assume that received delta throughput values should closely approximate received reverse feedback values.

The only apparent solution to sender side liar detection is for the sender's most immediate leaf router to artificially limit the senders congestion window size and consequently their send rate by putting a known, low value into the delta throughput of that senders outgoing packets. Once any router on an XCP flow's end-to-end path has entered a value into delta throughput that value can only be lowered, since the bottleneck router with the most congestion or lowest overall bandwidth must be able to throttle incoming traffic. By assuming the role of the bottleneck temporarily, the sender's leaf router can monitor to see if any reverse feedback values are returned with higher than expected values; any receiver who is observed reporting these erroneous values can be considered an attacker. The side effects of this solution are that in order to make these assessments a sender's bandwidth must be artificially capped at various intervals, reducing overall performance for legitimate traffic. It remains to be seen whether this method can successfully be applied in a probabilistic manner to minimize router overhead and collateral detrimental effects.

C. Conclusions

We confirm via detailed simulations that it is possible to create highly successful, malicious, cheating XCP clients[2]. The existence of receiver side cheating means that even conforming XCP implementations are vulnerable to exploitation. The feedback mechanisms that make XCP a strong protocol become its greatest weakness when confronted with unresponsive flows backed by fabricated feedback, allowing attackers to acquire almost total control of available bandwidth resources. Given the potential gains available to unscrupulous users on an XCP enabled network, the incentive to cheat is very high.

Although attacking effectively and probing for bandwidth does induce some packet loss overhead, the numbers of drops still remain far less than that for typical TCP flows, meaning that the actual negative ramifications of this overhead are negligible. Even when multiple attackers are present on the same network links, overall utilization remains almost as high as if the flows were cooperating normally, adding further incentive for this practice.

Acknowledgments

We wish to thank the anonymous reviewers for their helpful feedback. We also gratefully acknowledge support by DARPA under the Control Plane Program BAA04-11, and the NSF under CAREER Award #0546216.

References

[1] D. Katabi, M. Handley, and C. Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks," in *Proc. of SIGCOMM*, August 2002.

[2] A. Falk, Y. Pryadkin, and D. Katabi, "Specification for the Explicit Control Protocol (XCP)," draft, November 2006, http://www.isi.edu/isi-xcp/docs/draft-falk-xcp-spec-02.txt.

[3] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson, "Tcp congestion control with a misbehaving receiver," *ACM SIGCOMM Computer Communications Review*, vol. 29, no. 5, pp. 71–78, October 1999.

[4] R. Sherwood, B. Bhattacharjee, and R. Braud, "Misbehaving tcp receivers can cause internet-wide congestion collapse," in *Proc. of CCS*, Alexandria, VA, November 2005.

[5] K. K. Ramakrishnan and R. Jain, "A binary feedback scheme for congestion avoidance in computer networks," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 158–181, May 1990.

[6] K. K. Ramakrishnan, S. Floyd, and D. Black, *The Addition of Explicit Congestion Notification (ECN) to IP*, Sept. 2001, rFC 3168, Proposed Standard.

[7] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman, "One more bit is enough," in *Proc. of SIGCOMM*, Philadelphia, PA, August 2005.

[8] M. Georg and S. Gorinsky, "Protecting tfrc from a selfish receiver," in *Proc. of ICAS-ICNS*, Oct. 2005.

[9] S. Low, L. Andrew, and B. Wydrowski, "Understanding XCP: Equilibrium and fairness," in *Proc. of INFOCOM*, Barcelona, Spain, March 2005.

[10] Y. Zhang and M. Ahmed, "A control theoretic analysis of xcp," in *Proc. of IEEE Global Internet Symposium*, Miami, FL, March 2005.

[11] Y. Zhang and T. Henderson, "An implementation and experimental study of the explicit control protocol (xcp)." in *Proc. of INFOCOM*, Miami, FL, March 2005.

[12] L. Garber, "Denial-of-service attacks rip the internet," *IEEE Computer*, vol. 33, no. 4, pp. 12–17, April 2000.

[13] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms," *ACM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, April 2004.

[14] R. Kapoor, L.-J. Chen, L. Lao, M. Gerla, and M. Y. Sanadidi, "Capprobe: A simple technique to measure path capacity," in *Proc. of SIGCOMM*, Portland, OR, September 2004.

[15] T. Anderson, A. Collins, A. Krishnamurthy, and J. Zahorjan, "Pcp: Efficient endpoint congestion control," in *Proc. of NSDI*, May 2006.

[16] "The network simulator ns-2." http://www.isi.edu/nsnam/ns.

[17] "Cert advisory ca-1998-01 smurf ip denial-of-service attacks," http://www.cert.org/advisories/CA-1998-01.html, 1998.

[18] D. Katabi, "Xcp's performance in the presence of malicious flows," in *Proc. of Workshop on Protocols for Fast Long Distance Networks*, February 2004.

[19] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proc. of OSDI*, December 2002.

[2] We are currently evaluating the effectiveness of attacks on the Linux XCP implementation [11] using the Emulab network testbed [19].