CRAFT: A New Secure Congestion Control Architecture

Dongho Kim †, Jerry T. Chiang †, Yih-Chun Hu †, Adrian Perrig ‡, and P. R. Kumar †

†Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, IL, USA
‡Department of Electrical and Computer Engineering, Carnegie Mellon University, PA, USA
†{dkim99,chiang2,yihchun,prkumar}@illinois.edu, ‡perrig@cmu.edu

ABSTRACT

Congestion control algorithms seek to optimally utilize network resources by allocating a certain rate for each user. However, malicious clients can disregard the congestion control algorithms implemented at the clients and induce congestion at bottleneck links. Thus, in an adversarial environment, the network must enforce the congestion control algorithm in order to attain the optimal network utilization offered by the algorithm.

Prior work protects only a single link incident on the enforcement router, neglecting damage inflicted upon other downstream links. We present CRAFT, a capability-based scheme to secure *all* downstream links of a deploying router. Our goal is to enforce a network-wide congestion control algorithm on all flows. As a reference design, we develop techniques to enforce the TCP congestion control. Our design regulates all flows to share bandwidth resources in a TCP-fair manner by emulating the TCP state machine in a CRAFT router. As a result, once a flow passes a single CRAFT router, it is TCP-fair on all downstream links of that router.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

General Terms

Security

Keywords

TCP, Congestion Control

1. INTRODUCTION

The congestion control problem of fairly distributing network resources among users is a long-standing problem in networking research community. While most work in congestion control assumes that all entities follow the rules specified by a congestion control algorithm, some work has also considered an adversarial environment where a misbehaving user deviates from the specified rule [10, 11], thereby gaining network resource allocation that exceeds his fair share or suppressing the network resource allocated to others.

Some work adopts a reactive approach to defend against misbehaving users. Floyd et al. [4] use a TCP throughput equation to determine the proper throughput for a flow and categorize any flow using more than its fair throughput as "not TCP friendly". Stoica et al. [12] estimate the rate of a flow and assign that flow a fair share of a link according to the estimated rate. In contrast to the protocols proposed by Floyd et al. and Stoica et al., fair queueing [3] is a preventive approach where a router allocates a fixed amount of bandwidth to a flow or the aggregate of several flows when that router experiences congestion.

CCS'10, October 4–8, 2010, Chicago, Illinois, USA.
ACM 978-1-4503-0244-9/10/10.

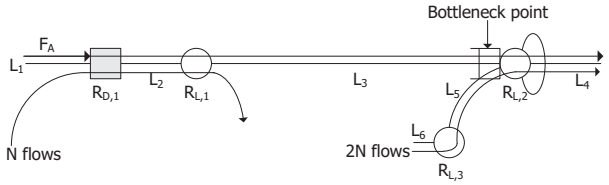


Figure 1: An example that illustrates single-link protection is insufficient for protecting downstream links.

Problem. We consider the purpose of these schemes to be fair sharing of network bandwidth. One might view congestion control as a scheme that rate-limits traffic before congestion happens to avoid wasting network bandwidth. The fairness of congestion control algorithm can be formalized using an optimization framework [7]. In this context, past work for securing fair rate allocation provides desirable properties for the fair sharing of the link immediately behind a deployed router. However, these schemes have a limitation in early phases of deployment where some routers may not have yet deployed the scheme. In such situations, the link behind a legacy router can be prone to congestion. Since different links have a different number of traversing flows and different amount of link bandwidth, the fair share on a deployed link is not necessarily the fair share on a downstream legacy link. This limitation motivates us to study a *minimal deployment* scheme that securely provides fair sharing of downstream links.

Illustration. We illustrate how a single-link protection mechanism is not sufficient to enforce that every flow in the network fairly shares network bandwidth at its bottleneck link. Figure 1 shows a network in which all links have the same link capacity and some routers do not deploy any protection mechanism. In this example, we use the concept of *fairness* as equal share of bandwidth. Let there be a flow that traverses multiple links together with different number of flows at each link. The bandwidth-fair rate of a flow traversing a link is simply the bandwidth of the link divided by the number of flows sharing the link. In this example network, flow F_A and N other flows traverse router $R_{D,1}$ that deploys a single-link protection mechanism, enforcing that each flow gets $\frac{1}{N+1}$ of the link capacity. That is, F_A shares link L_2 with N other flows in a bandwidth-fair manner. Suppose F_A's destination is different from

that of the other N flows; then F_A, but not any of the N other flows, would traverse the legacy router $R_{L,2}$. This legacy router does not employ any fairness-guaranteeing schemes. If $2N$ other flows also traverse through the router $R_{L,2}$, F_A could obtain roughly twice as much bandwidth as each of the other $2N$ flows could.

Challenge. One possible solution of this problem is letting a deployed router obtain information about the status of a bottleneck link and adjust flow rates accordingly. However, it is difficult to estimate the proper fair share of bottleneck link bandwidth. Though the router incident to a bottleneck link can deliver enough information for a deployed router to adjust flow rates, this approach requires the router incident to the bottleneck link to do some work. In other words, it still requires some level of deployment.

Our approach. To provide *network-wide* protection against misbehaving flows, we securely enforce TCP behavior by emulating the TCP state machine of each flow. The rationale behind enforcing TCP is that TCP is an end-to-end protocol that provides network-wide fair rate. Since TCP is end-to-end, it treats the network as a black box and does not require any information about the fair share of a bottleneck link. Different from our example, our goal of enforcing TCP fairness is not to assign equal rates since TCP rate of each flow depends on several factors, such as round trip time. There is theoretical work [9] that analyzes the fairness of TCP using optimization-based framework and obtains a utility function optimized by TCP.

In our poster, we present CRAFT (Capability-based Regulation of All Flows and Traffic), a secure congestion control architecture that provides network-wide fairness in a partially-deployed network.

2. PROTOCOL

In this section, we present how a CRAFT router can securely emulate the TCP state machine of a traversing flow. To better understand the concept of our approach, we first present a strawman design. Our strawman design illustrates a method that emulates the TCP congestion control protocol [1] with high overhead in an idealized environment where all packets reach the destination in order. We provide careful treatment on how CRAFT relaxes these limitations in Section 2.2.

2.1 Strawman Design

The TCP congestion control assumes that a pair of sender and receiver would behave in a manner specified in the protocol; specifically, the receiver honestly acknowledges receiving a packet and the sender increases the rate of the flow as specified by TCP upon receiving that acknowledgement. However, in an adversarial environment, a sender can arbitrarily increase its rate without having received any acknowledgments. Furthermore, a receiver can send acknowledgments without actually having received a packet [10] since TCP acknowledgments are not cryptographically dependent on the data in the TCP packet. Our strawman design illustrates how we can prevent a TCP flow from misbehaving.

Intuition. The main objective of our strawman design is to ensure that a downstream packet is indeed received by the receiver and the acknowledgment is indeed received by the sender. A strawman router can generate a Random Value for each packet to verify that the receiver has received a downstream packet, that the receiver has sent the Random Value back to the sender in its acknowledgment packet, and that the sender has received that acknowledgment.

When the strawman router sees a downstream packet, it inserts a Random Value into the packet, routes the packet to the receiver, and stores the Random Value and its associated flow in the router's memory. The receiver, upon receiving the packet, sends the Random Value back to the sender along with the TCP acknowledgment. The next time the sender wishes to send a packet to the receiver, the sender includes in the packet the Random Value he received that was embedded in the acknowledgment of the previous packet.

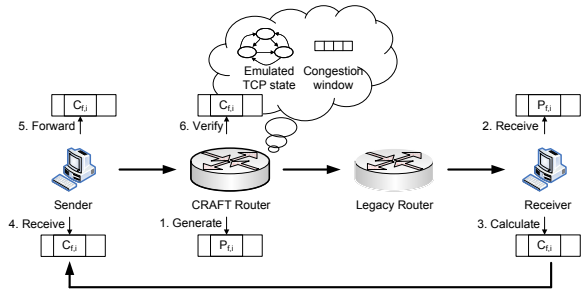


Figure 2: Capability-based enforcement of TCP congestion control algorithm. A CRAFT router generates a pre-capability ($P_{f,i}$) for a packet (i) of a flow (f). After the receiver gets the pre-capability, the receiver calculates and forwards a new capability ($C_{f,i}$) to the sender. The sender includes received capability to the CRAFT router in a future packet.

Since we assumed that the path from the sender to the receiver does not change, the data packet including the Random Value will reach the strawman router. The strawman router then verifies that the Random Value included in the packet is the same as that stored in the router. Since the Random Value is generated by the strawman router, a matching pair of Random Values in the packet and in the router's local memory implies that the previous packet was successfully received and acknowledged. In other words, the receiver and the sender collaboratively acknowledge to the router that the Random Value and the packet therein were successfully received.

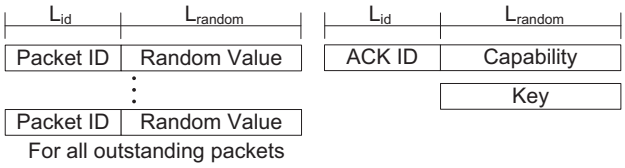
This architecture addresses the *minimal trust* since a strawman router is only required to trust itself. Our strawman design also addresses the *asymmetry of Internet path* since our strawman router does not require upstream packets or acknowledgment packets to traverse the same strawman router that the downstream packets traversed. Our CRAFT protocol preserves these properties.

2.2 CRAFT

Since a strawman router needs to maintain a Random Value for each outstanding packet, the required memory space of a strawman router can be huge with a large number of outstanding packets. CRAFT relaxes this limitation to efficiently emulate the TCP state of each flow in the real Internet environment. We do not focus on compliance of any particular TCP variant, but instead, on providing techniques to efficiently handle reordering and loss as specified by the standard document [1]. Since we design CRAFT without any specific TCP variants in mind, CRAFT emulates the congestion window (cwnd) of a flow using the maximum allowed cwnd based on the standard. Usually, TCP variants differ only in the management of cwnd in response to packet losses. Thus, by properly modifying our proposed techniques in emulating the cwnd, CRAFT can be readily adapted for specific subsets of most TCP variants.

The overview of CRAFT operation is illustrated in Figure 2. Due to space constraint, we focus on how we conserve the memory of router while keeping security. We briefly sketch how to calculate the cwnd for a flow and how to handle packet loss and reordering.

The inefficient use of memory space in our strawman design results from the fact that a Random Value inserted into a packet corresponds to only a single packet and the Random Value for each outstanding packet should be stored for verification. We thus let a



(a) Strawman design (b) CRAFT

Figure 3: Comparison of router memory usage between strawman design and CRAFT for a flow: L_{id} and L_{random} are reference length for comparison

CRAFT router insert into each packet a secret *pre-capability*, which is calculated using the Packet ID. We then define the *capability* of a flow to be an aggregate of all the past pre-capabilities inserted into its packets.

Since the pre-capability of a packet is calculated using its Packet ID, a CRAFT router does not need to store all prior pre-capabilities issued to a flow, thereby conserving the limited memory space of a router. However, CRAFT needs to provide unpredictability for pre-capability and capability as Random Value does. CRAFT provides unpredictability by utilizing computationally efficient keyed MAC (Message Authentication Code). A CRAFT router uses its secret key K and a cryptographically secure hash function g to derive a pre-capability $P_{f,i}$: $P_{f,i} = g(K, f, i)$, where f is the flow ID, and i is the Packet ID. Flow ID is a large and unique value randomly generated by the router when the flow is created.

We further elaborate on the g function with a telescoping construction: $g(K, f, i) = E_K(f||i) \oplus E_K(f||i+1)$ where E_K represents a computationally efficient keyed MAC such as HMAC [8] and $||$ is the concatenation operator. The telescoping construction enables the CRAFT router to calculate a capability corresponding to a set of contiguous pre-capabilities in constant time. If E_K is a hash function in a Random Oracle Model [2], then $E_K(f||i)$ is indistinguishable from a random number because the pair (f, i) has not been seen before. Since K is known only to the CRAFT router, it is computationally inefficient to guess $E_K(f||i)$. Moreover, $P_{f,i}$ is also indistinguishable from a random number.

When the receiver receives a packet (j) with a pre-capability ($P_{f,j}$), the receiver constructs the capability ($C_{f,j}$) associated with the packet. A capability ($C_{f,j}$) is defined to be the exclusive-or of all received pre-capabilities up to j: $C_{f,j} = P_{f,0} \oplus \ldots \oplus P_{f,j}$. By using the exclusive-or function, we can take advantage of a desirable property: if any input and its distribution are unknown, the output is uniformly distributed over the domain, yielding the largest uncertainty and secrecy in the information theoretic sense. Moreover, since $P_{f,j} = E_K(f||j) \oplus E_K(f||j+1)$, the capability can be calculated efficiently using $C_{f,j} = E_K(f||0) \oplus E_K(f||j+1)$. Hence, we need only a single xor operation to calculate the capability associated with a set of contiguous pre-capabilities.

In the above example, capability $C_{f,j}$ proves that all packets from Packet ID 0 to j are received by a receiver. This capability and j are delivered to the sender and are used to prove the receipt of packets to the CRAFT router. We call j as ACK ID. When CRAFT router receives $C_{f,j}$ and j, it verifies the received capability. If the received capability is valid, CRAFT router increases the cwnd for the corresponding flow following the TCP standard.

Our construction of pre-capability and capability eliminates the need to store a Random Value for each outstanding packet. As shown in Figure 3(b), the router only needs fixed size of space.

When a network experiences congestion, a pre-capability may be lost. Then, a contiguous capability $C_{f,j} = P_{f,0} \oplus \ldots \oplus P_{f,j}$ cannot be constructed. To handle this loss, the receiver discloses the loss to the sender, and the sender discloses that loss together with the rest of non-contiguous capability to CRAFT router. For example, if a pre-capability for packet ID $k(< j)$ is lost, corresponding capability is $C_{f,j} = P_{f,0} \oplus \ldots \oplus P_{f,k-1} \oplus P_{f,k+1} \oplus \ldots \oplus P_{f,j}$. CRAFT router verifies the received non-contiguous capability taking the disclosed loss into consideration.

3. LIMITATIONS

Per-Host Fairness. CRAFT considers per-flow fairness since it emulates the state machine of a TCP flow. To use CRAFT to enforce host-fairness, a scheme to guarantee a fair share of the number of connections among hosts is necessary. Without such a scheme, an attacker can initiate an excessive number of flows.

Per-Flow Operation. CRAFT achieves space-efficiency at the expense of hash calculation. If the computation overhead is significant, CRAFT may not be practical. However, a recent software router implementation [5] achieves about 10 Gbps throughput for forwarding 64 byte packets with IPsec operation. We believe that an ASIC implementation can further improve the throughput.

Vulnerability of TCP. CRAFT prevents a flow from deviating from the flow rate specified by the TCP standard. There can be other attacks against TCP [6, 10]. Our goal is to enforce TCP behavior, not to fix all possible vulnerabilities of the TCP congestion control.

4. POSTER ORGANIZATION

Due to space constraint, we do not present our evaluation results in this proposal. In our poster, we will present evaluation results for the overhead and the effectiveness of CRAFT as well as motivation and basic architecture.

5. REFERENCES

[1] M. Allman, V. Paxson, and W. Stevens. TCP congestion control. RFC 2581, April 1999.

[2] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of CCS '93*, pages 62–73, New York, NY, USA, 1993. ACM.

[3] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *In Proceedings of SIGCOMM '89*, pages 1–12, New York, NY, USA, 1989. ACM.

[4] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the internet. *Networking, IEEE/ACM Transactions on*, 7(4):458 –472, Aug 1999.

[5] S. Han, K. Jang, K. Park, and S. Moon. Packetshader: a GPU-accelerated software router. *SIGCOMM 2010*.

[6] A. Herzberg and H. Shulman. Stealth dos attacks on secure channels. In *Proceedings of NDSS '10*, 2010.

[7] F. Kelly, A. Maulloo, and D. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. In *Journal of the Operational Research Society*, volume 49, 1998.

[8] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication. RFC 2104, February 1997.

[9] S. H. Low and D. E. Lapsley. Optimization flow control. i. basic algorithm and convergence. *Networking, IEEE/ACM Transactions on*, 7(6):861 –874, Dec. 1999.

[10] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP congestion control with a misbehaving receiver. *SIGCOMM CCR*, 29(5):71–78, 1999.

[11] R. Sherwood, B. Bhattacharjee, and R. Braud. Misbehaving TCP receivers can cause Internet-wide congestion collapse. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, pages 383–392, New York, NY, USA, 2005. ACM.

[12] I. Stoica, H. Zhang, and S. Shenker. Self-verifying CSFQ. In *Proceedings of INFOCOM '02*, volume 1, pages 21 – 30 vol.1, 2002.