Dealing with Receiver Misbehavior in Multicast Congestion Control

Vijay Arya, Thierry Turletti

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Dealing with Receiver Misbehavior in Multicast Congestion Control

Vijay Arya — Thierry Turletti

N° 4899

Juillet 2003

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--4899--FR+ENG

Dealing with Receiver Misbehavior in Multicast Congestion Control

Vijay Arya , Thierry Turletti

Thème 1 — Réseaux et systèmes
Projet Planete

Rapport de recherche n° 4899 — Juillet 2003 — 23 pages

Abstract: In multicast congestion control, receivers can misbehave by maliciously causing congestion to steal network bandwidth from well behaved flows. In source driven congestion control protocols (SDCC), receivers misbehave by sending a wrong feedback to the source. In receiver-driven congestion control protocols(RDCC), receivers misbehave by inflating their subscriptions. In this report, we present techniques to deal with these misbehaviours. Firstly, we show that when network tomography tools such as MINC are used in conjunction with SDCC protocols, they can aid in misbehavior detection. But in order to use MINC for misbehaviour detection, MINC itself must be made immune to misbehaviour. we analyze the effect of misbehavior within MINC and propose two techniques to detect and prevent misbehavior in MINC. For RDCC protocols, a misbehaviour prevention mechanism based on in-band distribution of keys was recently proposed for FLID-DL protocol. In this report, we show how we can design keys which can prevent misbehaviour in most RDCC protocols.

Key-words: MINC, Network Tomography, Misbehavior, Multicast Congestion Control, secret sharing.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Gestion du mauvais comportement des récepteurs dans les Protocoles de Contrôle de Congestion Multipoint

Résumé : Les protocoles de contrôle de congestion actuels peuvent permettre à des récepteurs peu scrupuleux de mal se conduire en accaparant plus de bande passante qu'ils ne devraient. Avec les protocoles de contrôle de congestion contrôlés par la source (SDCC), il est possible à des récepteurs malveillants de falsifier les rapports de réception qu'ils envoeint à la source. En ce qui concerne les protocoles de contrôle de congestion contrôlés par les récepteurs (RDCC), certains récepteurs peuvent décider de s'abonner à toutes les couches de la source sans tenir compte des débits qui leur sont alloués. Dans ce rapport, nous présentons des techniques pour prévenir ces comportements indésirables. Nous montrons comment un outil de tomographie de réseau comme MINC peut être utilisé pour détecter ces comportements malveillants avec les protocoles SDCC. L'outil de tomographie doit dans un premier temps être rendu insensible à ces attaques. Nous analysons le comportement de MINC et proposons des mécanismes de détection et prévention contre ces comportements malicieux. Pour les protocoles RDCC, un mécanisme de prévention de comportement malicieux basé sur la distribution de clés a été proposé récemment pour le protocole FLID-DL. Nous proposons un nouveau mécanisme de génération de clés pour la plupart des protocoles RDCC.

Mots-clés : Attaques, Contrôle de Congestion Multipoint, MINC, Outils de Tomographie du Réseau, partage de secrète.

1 Problem

Congestion related misbehavior is possible in a network due to the presence of both types of users (flows) - those who respond to congestion and those that do not. Malicious users who do not respond to congestion are able to unjustfully steal network bandwidth from those users who do respond to congestion. For this, malicious users *intentionally* cause congestion for a certain duration of time forcing well behaved network flows to back-off and then claim the bandwidth left over by the well behaved flows. In this manner, by bearing the congestion they have caused for some time, they steal network bandwidth from well behaved flows. In the context of multicast congestion control, receivers of a multicast group can misbehave to achieve such an effect. By meddling with the multicast congestion control mechanism, misbehaving receivers can obtain a rate which they desire instead of the admissible rate which the present network conditions offer them [11]. The importance of multicast congestion control cannot be over emphasized [9].

In multicast congestion control, receivers estimate the capacity available to them in the network, generally based on congestion losses. How this estimate is utilized for congestion control depends on two paradigms. In sender or *Source-Driven Congestion Control* (SDCC), each multicast receiver transmits a feedback report to the sender. The sender then calculates the rate at which to inject packets into the network as an aggregate function of feedback reports received from all the receivers. In *Receiver-Driven Congestion Control* (RDCC), the sender either splits or replicates the data into several different groups, each with a different sending rate, which together constitute a multicast session. Each receiver subscribes to one or more groups based on its available network capacity and responds to congestion by unsubscribing to groups. Both these approaches are vulnerable to receiver misbehavior, the motivation being to receive the rate desired by the receiver instead of the rate offered by the present network conditions. A misbehaving SDCC receiver sends a wrong feedback which causes the sending rate of the sender to increase resulting in congestion. A misbehaving RDCC receiver unjustfully subscribes to more groups and inflates its subscription to cause congestion. Congestion forces the well behaved flows to reduce their sending rate, leaving more bandwidth for the misbehaving receivers.

2 MINC and Misbehavior

Network tomography allows us to obtain the internal characteristics of networks based on end-to-end measurements. *Multicast-based Inference of Network internal Characteristics* (MINC) [4, 1] is a tomography tool which allows us to infer the internal characteristics of the network underlying a multicast tree. Based on end-to-end measurements between the source and the receivers of a multicast group, MINC can infer the loss rates and delay characteristics of links (nodes) and the multicast tree topology. For the loss rates, the source injects probe packets into the multicast tree and each receiver reports whether it received the probe packet (`1`) or not (`0`). Based on these binary traces collected from receivers, loss rates of links are inferred. The probe packets can be the data packets of the transport protocol used to transfer data in the multicast tree itself, such as Real-time Transport Protocol(RTP) [17, 5]. The loss rates of links inferred by MINC essentially provide us with the "congestion tomography" of a multicast tree. Inference of such congestion tomography can

help to trap multicast receivers who maliciously cause congestion. When a multicast receiver steals network bandwidth by maliciously causing congestion, it causes an increase of loss rates on certain links in the logical multicast tree which can be seen via the congestion tomography constructed by MINC. If MINC is used in conjunction with multicast congestion control, it can prove to be useful in detecting misbehavior. However, a receiver who is maliciously causing congestion can cheat by reporting a false MINC feedback too, thus "hiding" the effect of its misbehavior from MINC. Consequently, misbehavior needs to be detected or prevented within MINC also. Not just for congestion misbehavior detection, even if MINC has to be used for any vital or fault-finding application, we believe it itself must be made immune to any kind of malicious attack.

In SDCC protocols, the sender maintains a single rate for the entire multicast group. Each multicast receiver transmits a feedback report to the sender which then calculates the rate at which to inject packets into the network based on the feedback reports received. A receiver may cheat by sending a wrong feedback. To detect receiver misbehavior in SDCC protocols, the correct congestion tomography inferred by MINC can be correlated with the false congestion feedback from misbehaving receivers or with the sending rate of the sender. When a receiver asks the sender to increase its sending rate in spite of high loss rates indicated by MINC, we can conclude that it is trying to cheat.

In RDCC protocols, if a receiver inflates its subscriptions, MINC shows high loss rates, but we can not be sure if any (misbehaving) receiver is responsible for this unless we correlate it with the number of layers the receiver has subscribed to. But this is not known to the sender. In the absence of any feedback from receivers, the sender generally does not change the sending rate of different layers. Thus in RDCC protocols, it can be inferred that congestion is occurring on certain links in the multicast tree and that there is something wrong, but it cannot be concluded whether certain external flows or a misbehaving receiver is responsible for this congestion. Hence, the congestion tomography is more suitable for misbehavior detection in protocols which use the SDCC approach.

3 Background and Related work

The problem of misbehavior in multicast congestion control was reported recently by Sergey, et al in [11], who showed that most multicast congestion control protocols are vulnerable to misbehavior. SDCC protocols, which use a single rate approach are suited for small sets of receivers and when the network is less heterogeneous in terms of link bandwidths. Protocols such as TFMCC [22], RMTP [14], PGMCC [16] and SAMM [2] use this approach. RDCC protocols which use multi-rate approach are suitable for heterogeneous networks. Protocols such as RLM [12], RLC [20], FLID-DL [3] use this approach. There also exist several hybrid protocols such as MLDA [19], SARC [21] and DSG [6] which use a combination of these approaches. We briefly describe TFMCC and FLID-DL protocols and show how misbehavior is possible in these protocols.

In TFMCC [22], each receiver calculates its TCP friendly rate based on the losses and RTT that it experiences. Receivers report their rate directly to the sender at random intervals of time to prevent feedback implosion. The slowest receiver becomes the CLR (current limiting receiver), continuously reports its rate and controls the sending rate of the sender until another receiver reports a lower rate. Assuming that receivers indulge in self-beneficial attacks, the following misbehavior is possible. If

the slowest receiver is behind a bottleneck link shared by several well-behaving or TCP flows, it can start reporting a false high rate causing the sender to raise the sending rate, resulting in congestion on the bottleneck. Thus the TCP flows reduce their sending rate leaving extra bandwidth for the malicious receiver. In this manner the slowest receiver can raise the sending rate of the sender up to the rate of the slowest well behaved receiver.

In FLID-DL [3], the sender encodes data into cumulative layers, each of which is made available on a separate multicast group. Receivers subscribe to layers in a cumulative manner and unsubscribe to them when they experience a congestion loss. The sender partitions the time into slots of duration T seconds each. If a receiver which has subscribed to layers $0..i$, experiences a packets loss during a time slot, it reduces its subscription level by one at the end of the time slot. That is, it unsubscibes to layer i, reducing it subscription level to $i - 1$. To coordinate subscription actions of receivers, the sender sends an increase signal into each packet of a time slot. If a receiver with subscription level i receives an increase signal j where $j \geq i$ and receives all its packets during the current time slot, it increases its subscription level by one during the next time slot. Assuming self-beneficial attacks, the following misbehavior is possible. Consider a receiver behind a bottleneck link shared by several well behaved flows. Such a receiver intentionally inflates its subscription level and causes congestion on the bottleneck, forcing the well-behaved flows to reduce their sending rate. As the congestion period recedes, the misbehaving receiver starts to obtain better quality data. In this manner the misbehaving receiver can steal network bandwidth from well behaved flows. By doing so, the misbehaving receiver may also be harming other users in the same multicast session since it is subscribing to layers in an uncoordinated manner.

Receiver misbehavior can be dealt with using either *misbehavior detection* or *misbehavior prevention*. The goal of misbehavior detection is to develop a procedure to reliably infer if a multicast receiver is misbehaving by providing a wrong feedback. On the other hand, the goal of misbehavior prevention is to design a mechanism which automatically prevents any multicast receiver from misbehaving. Misbehavior detection is a more difficult problem to solve than misbehavior prevention (as will become clear later). But the later generally involves change of the protocol code at the sender and receivers or modification of the system to incorporate the prevention technique which is avoided in the former.

Recently, sergey, et al [10] proposed a misbehavior prevention mechanism for RDCC protocols, which particularly suits the FLID-DL algorithm. Their prevention mechanism is based on the in-band distribution of keys which guard the access to multicast groups. These keys are lost when a receiver inflates its subscriptions to cause congestion. Thus the receiver is automatically prevented from increasing its subscription level when congestion occurs. We discuss this later in section 6.

In this report, we firstly show that when MINC is used in conjunction with SDCC protocols, it can aid in misbehavior detection. But for this, MINC itself must be first made immune to misbehavior. We analyze the effect of misbehavior within MINC and present techniques to deal with misbehavior within MINC. For RDCC protocols, we show how to design keys to prevent misbehavior. The balance of this report is organized as follows. Section IV describes the fundamentals of MINC and shows that the detection of multicast congestion misbehavior is possible with MINC. Section V analyzes the effects of misbehavior within MINC and subsequently presents misbehavior detection and prevention mechanisms for MINC. In section VI, we discuss misbehavior prevention

in RDCC protocols and show the design of a key which can be used in most RDCC protocols. We conclude with section VII.

4 Misbehavior detection in SDCC with MINC

4.1 MINC Fundamentals

MINC infers the internal characteristics of a network underlying a multicast tree by exploiting the inherent correlation in multicast traffic. We focus here on explaining how the loss rates or the passage rates of links are inferred by MINC since we use them extensively in this paper. Details of MINC can be found in [1]. Consider a simple multicast tree shown in figure 1 with source S, two

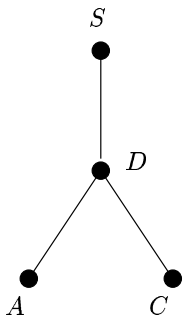

Figure 1: Multicast tree with two receivers

receivers A,C and the branching node D. Suppose that the source sends a stream of packets and each receiver observes whether it receives a packet (`1`) or not (`0`). Consider now, those packets which were received by C. These packets must have definitely crossed D (or the shared link SD) since they were received by C. Thus each of these packets was also sent on link DA but either crossed or was lost on this link. Thus the ratio of the number of packets which both A and C received to the number packets which C received gives the passage probability of link DA. These ideas are extended in MINC to calculate the passage probability of all links in the multicast tree. These passage probabilities can also help in inferring the topology of the multicast tree [15, 7].

4.2 Example of MINC with TFMCC protocol

Figure 2 shows the topology we used for simulation of TFMCC using `NS2`, with sender S and the slowest receiver C behind a bottleneck link shared with TCP flows. After 60 seconds, receiver C begins to cheat by reporting a false high rate. This is shown in figure 3. We used TFMCC packets to act as probes and traced the packets received by each receiver and gave it to MINC inference engine in batches of 500 trace feedbacks. Figure 4 shows the passage probability of the bottleneck link calculated by MINC. Thus by correlating the sending rate of the sender along with the MINC loss

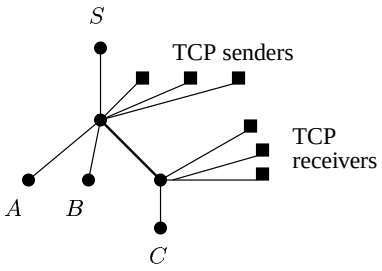

Figure 2: Topology for simulation

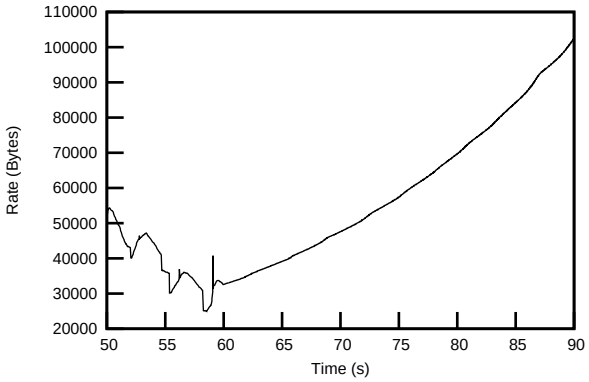

Figure 3: C cheats by reporting false rates

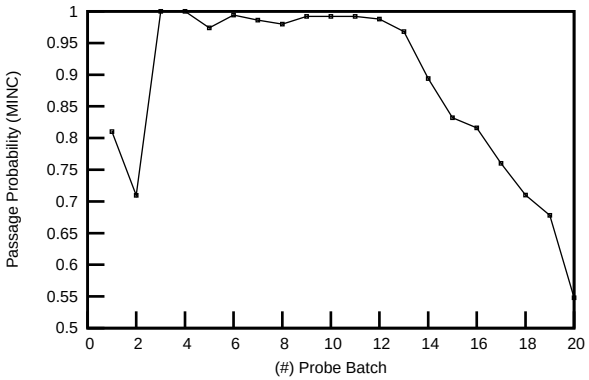

Figure 4: Passage probability of path DC

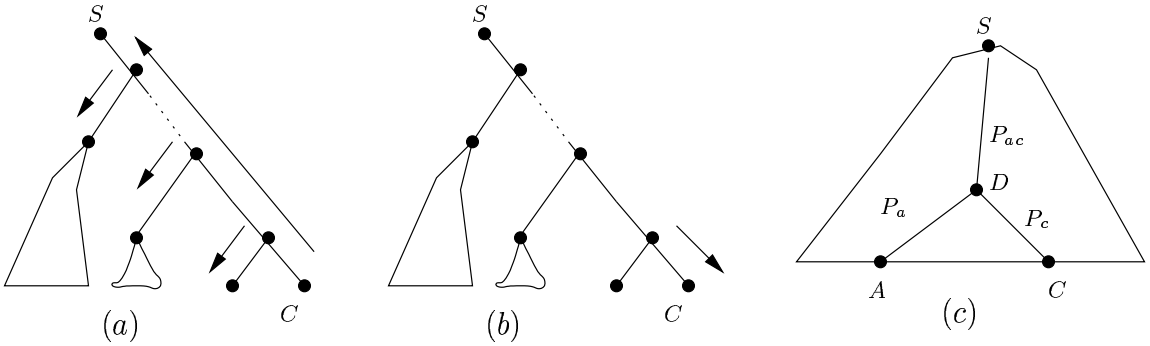

Figure 5: Effects of misbehavior on passage probabilities. (a) Receiver C always says Yes, (b) Receiver C always says No (c) Part of the multicast tree with 2 receivers

inference, misbehavior can be detected. But before being able to do this, MINC itself must be made immune to misbehavior.

5 Handling Misbehavior in MINC

5.1 Problem

We now focus on the problem of misbehavior in MINC related to inference of loss rates. The sender injects n probe packets into the multicast tree and each receiver reports whether it received the probe packet (`1`) or not (`0`). Based on the $n \times r$ binary matrix, the MINC procedure tries to infer the passage probabilities[1] of links(nodes). If a receiver misbehaves by reporting a wrong feedback, it distorts the passage probabilities inferred by MINC in the multicast tree. The next section explains how passage probabilities are estimated in MINC and how these estimates can be distorted by a misbehaving receiver. It also explains certain properties of these estimates which motivate our detection technique. The subsequent two sections describe how distortions in estimates can be detected to conclude misbehavior or how they can be prevented altogether.

5.2 Impact of cheating on passage probabilities

Figure 5(a) shows one possible impact of misbehavior on the passage probabilities calculated by MINC, when the receiver C tries to cheat by falsely reporting more 1's. Here, the receiver C cheats for an interval of feedbacks during which it changes all 0's to 1's. The passage probability of the path from C to the sender increases and the passage probability of any path from any other receiver to C's ancestor decreases. How the passage probability of an individual link on a path from any other receiver to an ancestor of C changes is shown in the figure (using arrow marks). Thus the

[1] *Passage probability* refers to the probability of a packet crossing a link and *Loss probability* refers to the probability of a packet getting lost on a link.

passage probabilities in a large region of the multicast tree change. Figure 5(b) shows a possible impact of misbehavior when the receiver C tries to cheat by falsely reporting more 0's. Here, the receiver C cheats for an interval of feedbacks during which it changes all 1's to 0's. In this case only the passage probability of the path from C to C's father changes to be congruous with data reported by C. The passage probabilities of the rest of the multicast tree remain unchanged. The rest of this section analyses the cause of such impacts.

Figure 5(c) shows the part of a multicast tree containing two receivers A and C with the sender S. Let F be the feedback array reported by receivers A and C for N probes. Each feedback $f \in \{(00),(01),(10),(11)\}$. Let n_{ij} be the total number of feedbacks of type (i,j), where $i,j \in \{0,1,* = \{1 \cup 0\}\}$. Each correct feedback stands for a packet sent by S which either crossed or was lost on a path. For example n_{01} are the total number of packets which crossed SD and DC but were lost on DA. Assuming *Bernoulli* losses, by *Maximum Likelihood Principle*, the passage probability on a path can be estimated by dividing the total packets which crossed a path by the total packets sent on that path. Now, observe that among n_{*1} packets sent on DA (after they crossed SD), only n_{11} actually crossed RA. Thus the passage probability of path DA (P_a) can be estimated by (1) and similarly the passage probability of path DC(P_c) by equation (2). Denote by P_{ac} the passage probability of path SD. (Compliments of P_a, P_c and P_{ac} are P'_a, P'_c and P'_{ac} respectively).

$$P_a = \frac{n_{11}}{n_{*1}}, \quad (P'_a = \frac{n_{01}}{n_{*1}}) \tag{1}$$

$$P_c = \frac{n_{11}}{n_{1*}}, \quad (P'_c = \frac{n_{10}}{n_{*1}}) \tag{2}$$

Having done this, observe that $(n_{11}/N) = P_{ac} \cdot P_a \cdot P_b$. Thus P_{ac} can be estimated using (3). Equivalently, P_{ac} could have been estimated using either n_{10}/N, n_{01}/N, or n_{00}/N. But all of them can be proven to be exactly equal to the right hand side of (3)

$$P_{ac} = \frac{n_{11}/N}{(n_{11}/n_{*1})(n_{11}/n_{1*})} = \frac{n_{*1}n_{1*}}{Nn_{11}} \tag{3}$$

Now if receiver C misbehaves by reporting a 1 instead of a 0, it can cause two types of transformations in the feedback array

$$x : (00) \Rightarrow (01) \quad y : (10) \Rightarrow (11)$$

Equations (4)-(6) show the impact on passage probabilities if receiver C either transformed n_x feedbacks using an x transformation, or n_y feedbacks using a y transformation.

$$P_a = \frac{n_{11}}{n_{*1} + n_x} \Downarrow \qquad P_a = \frac{n_{11} + n_y}{n_{*1} + n_y} \Uparrow \tag{4}$$

$$P_c = \frac{n_{11}}{n_{1*}} \Leftrightarrow \qquad P_c = \frac{n_{11} + n_y}{n_{1*}} \Uparrow \tag{5}$$

$$P_{ac} = \frac{(n_{*1} + n_x)n_{1*}}{Nn_{11}} \Uparrow \qquad P_{ac} = \frac{(n_{*1} + n_y)n_{1*}}{(n_{11} + n_y)N} \Downarrow \tag{6}$$

In general, x transformations increase($\Uparrow$) P_{ac} and decrease($\Downarrow$) P_a and y transformations decrease P_{ac} and increase P_a. The x transformation leaves P_c untouched and y increases P_c. The type of change in the passage probabilities P_{ac} and P_a by x and y transformations depends on the original passage probabilities, n_x and n_y as is shown in the following lemma.

Lemma 1 *If a receiver cheats by reporting 1 instead of 0 resulting in n_x transformations from $(00) \Rightarrow (01)$ and n_y transformations from $(10) \Rightarrow (11)$, then P_{ac} and P_a remain unchanged if $n_x/n_y = P'_a/P_a$.*

Proof After cheating we have,

$$P_{ac}^{cheat} = \frac{(n_{*1} + n_x + n_y)n_{1*}}{(n_{11} + n_y)N}$$

Now $P_{ac}^{cheat} = P_{ac}$ iff

$$\frac{(n_{*1} + n_x + n_y)n_{1*}}{(n_{11} + n_y)N} = \frac{n_{*1}n_{1*}}{n_{11}N}$$

ie., iff

$$\frac{n_x}{n_y} = \frac{n_{*1}}{n_{11}} - 1$$

ie., iff

$$\frac{n_x}{n_y} = \frac{n_{01}}{n_{11}} = \frac{P'_a}{P_a}$$

Similarly for P_a. ∎

If $P_a < 1/2$, then the increase in P_{ac} caused by one x transformation is less than the decrease caused by a one y transformation. Thus an equal number of x and y transformations result in total net decrease. If $P_a > 1/2$, then the increase in P_{ac} caused by one x is more than the decrease caused by one y resulting in a net increase for an equal number of x and y transformations. Similarly for P_a, if $P_a < 1/2$, then the decrease by x is less than the increase by y and, if $P_a > 1/2$, the decrease by x is more than the increase by y. For any value of P_a, if C cheats such that the ratio of x and y transformations is same as that of P'_a/P_a, the increase and decrease nullify each other resulting in no change in P_{ac} and P_a. Also, if C cheats such that it always causes an equal number of x and y transformations, it will fail create any change in passage probabilities only when $P_a = 1/2$ (other than for P_c).

Definition 1 *$MM(i \rightarrow i',\ I,\ p_{cheat})$, $i \in \{0, 1\}$ is a Misbehavior Mechanism in which a receiver cheats for an interval $I(\leq N)$ of feedbacks during which it changes every i to i' with probability p_{cheat}*

For example, in $MM(1 \rightarrow 0,\ I,\ 1)$, a receiver cheats for an interval I of feedbacks during which it changes all 1's to 0's.

Lemma 2 *If receiver C cheats using* $MM(0 \rightarrow 1,\ I,\ p_{cheat})$*, there is a high chance that it changes the passage probabilities* P_{ac} *and* P_a

Proof After cheating, the ratio of x transformations to y transformations is

$$\frac{n_x}{n_y} = \frac{I \cdot P(f = *0) \cdot p_{cheat} \cdot P(f = 00 | f = *0)}{I \cdot P(f = *0) \cdot p_{cheat} \cdot P(f = 10 | f = *0)} = \frac{n_{00}}{n_{10}}$$

Now from lemma 1, for no change we require $n_x/n_y = n_{01}/n_{11} = P'_a/P_a$. $n_{00}/n_{10} = n_{01}/n_{11}$ only if $P_{ac} = 1$. Thus when $P_{ac} \neq 1$, this kind of cheating causes a perceptible change. ∎

Lemma 2 explains why in the example of fig 5(a) there was a change in the passage probabilities when C cheated for an interval of feedbacks by changing every 0 to a 1 in the interval.

If a receiver cheats by reporting a 0 instead of 1, it causes reverse transformations

$$x' : (00) \Leftarrow (01) \quad y' : (10) \Leftarrow (11)$$

In general, x' increases P_a and decreases P_{ac} and y' decreases P_a and increases P_{ac}. x' leaves P_c untouched and y' decreases P_c. If a receiver causes $n_{x'}$ and $n_{y'}$ transformations in the feedback array, the conditions for no change of passage probabilities P_{ac} and P_a still remain same as in lemma 1.

$$\frac{(n_{*1} - n_{x'} - n_{y'})n_{1*}}{(n_{11} - n_{y'})N} = \frac{n_{*1}n_{1*}}{n_{11}N} \Rightarrow \frac{n_{x'}}{n_{y'}} = \frac{n_{01}}{n_{11}} \tag{7}$$

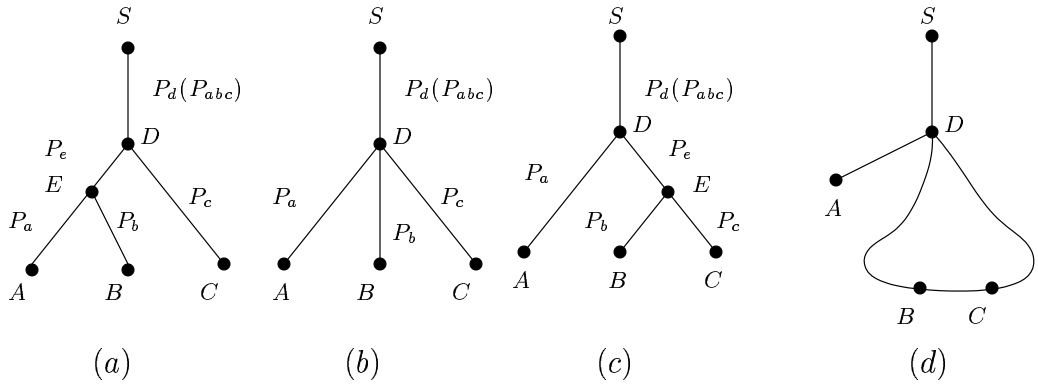

Figure 6: (a)-(c) Different topologies with 2 honest receivers and a suspect (d) Special topology

Lemma 3 *If receiver C cheats using* $MM(1 \rightarrow 0,\ I,\ 1)$*, it does not change the passage probabilities* P_{ac} *and* P_a*.*

Proof After cheating, the ratio of x transformations to y transformations is

$$\frac{n_{x'}}{n_{y'}} = \frac{I \cdot P(f = *1) \cdot p_{cheat} \cdot P(f = 01 | f = *1)}{I \cdot P(f = *1) \cdot p_{cheat} \cdot P(f = 11 | f = *1)} = \frac{n_{01}}{n_{11}}$$

which is the condition for no change from equation (7). ∎

Lemma 3 explains why there was no change in the passage probabilities in the example of figure 5(b). Now, if a receiver cheats by reporting both 1 to 0 and 0 to 1, then after cancellation of transformations, it is left with (n_x,n_y) or (n_x,$n_{y'}$) or ($n_{x'}$, n_y) or ($n_{x'}$, $n_{y'}$). x and y' together decrease P_a and increase P_{ac}. x' and y together increase P_a and decrease P_{ac}. y' decreases P_c and y increases P_c.

Although, whether or not P_{ac} and P_a change depends on the ratio n_x/n_y, the amount of change in P_{ac} and P_a depends on the given system. The net change (change/old value) in P_{ac} is given by

$$\mid \frac{n_{11}n_x - n_{01}n_y}{n_{*1}(n_{11} + n_y)} \mid$$

The maximum number of x and y transformations in a given system is limited by n_{00} and n_{10} respectively.

Now consider the case when we have 3 receivers A, B and C from a multicast tree. A and B are honest and C may cheat. Consider that the topology shown in Figure 6(a) is formed between these three receivers. Using the same terminology as for two receivers, let F be the feedback array for N probes, each element f of which $\in \{(000) \ldots (111)\}$. Let n_{ijk} be the total number of feedbacks of the form (i,j,k) where i,j,k $\in \{0, 1, * = \{0 \cup 1\}\}$. Firstly, observe that P_a and P_b can be determined without c's data since path EA can be probed using n_{*1*} and link EB can be probed using n_{1**}. Thus cheating by C effects only P_{abc},P_{ab}, and P_c.

Now, DC can be probed using one of the several ways to obtain P_c as shown in equation (8). $P_{abc} = P_{ac} = P_{bc}$ can be estimated using either A and C or B and C using (9) below and $P_e = P_{ab}/P_{abc}$.

$$\begin{aligned} P_c &= \frac{n_{011}}{n_{01*}} = \frac{n_{101}}{n_{10*}} = \frac{n_{111}}{n_{11*}} \\ &= \frac{n_{011} + n_{101} + n_{111}}{n_{01*} + n_{10*} + n_{11*}} \\ &= \frac{n_{1*1}}{n_{1**}} = \frac{n_{*11}}{n_{*1*}} \end{aligned} \tag{8}$$

$$\begin{aligned} P_{abc} &= P_{ac} = \frac{n_{**1}n_{1**}}{n_{1*1}N} \\ &= P_{bc} = \frac{n_{**1}n_{*1*}}{n_{*11}N} \end{aligned} \tag{9}$$

Lemma 4 *If all estimates of P_c and P_{abc} are equal before C cheats, they remain equal after C cheats using $MM(i \rightarrow i', I, p_{cheat})$.*

Proof We just prove that each estimate undergoes a corresponding proportionate change. For this proof, we assume $MM(0 \rightarrow 1, I, p_{cheat})$. The other case follows similarly. We prove that the last two estimates of P_c remain equal after cheating. This also proves that the estimates of P_{abc} remain equal after cheating. After cheating let the total number of feedbacks transformed from 1*0 to 1*1 be z_{1*1} and those transformed from *10 to *11 be z_{*11}. Now, z_{1*1} and z_{*11} can be expressed as below:

$$\begin{aligned} z_{1*1} &= I \cdot P(f = **0) \cdot p_{cheat} \cdot P(f = 1*0 | f = **0) \\ &= I \cdot p_{cheat} \cdot n_{1*0} = I p_{cheat}(n_{1**} - n_{1*0}) \\ z_{*11} &= I \cdot P(f = **0) \cdot p_{cheat} \cdot P(f = *10 | f = **0) \\ &= I \cdot p_{cheat} \cdot n_{*10} = I \cdot p_{cheat}(n_{*1*} - n_{*11}) \end{aligned}$$

After cheating the last two estimates in (8) become respectively,

$$\begin{aligned} \frac{n_{1*1} + z_{1*1}}{n_{1**}} &= \frac{n_{1*1} + I \cdot p_{cheat}(n_{1**} - n_{1*0})}{n_{1**}} \\ &= P_c + (1 - P_c) I \cdot p_{cheat} \\ \frac{n_{*11} + z_{*11}}{n_{*1*}} &= \frac{n_{*11} + I \cdot p_{cheat}(n_{*1*} - n_{*11})}{n_{*1*}} \\ &= P_c + (1 - P_c) I \cdot p_{cheat} \end{aligned}$$

∎

For the balance of this paper, we focus only on detection and prevention of 0 to 1 misbehavior as this is generally self beneficial as compared to 1 to 0 misbehavior.

5.3 Detection Overview

Our technique of detection is extremely simple and relies on the redundancy of information in the presence of several receivers. We assume that we have both honest and misbehaving receivers present in the multicast tree, and the goal of detection procedure being to prove if a given suspect receiver has cheated or not. Consider a group of sibling receivers(> 2) sharing a common path from the sender to their father. In order to estimate the passage probability on this common path, the feedback data of two siblings is sufficient, provided we have an adequate number of probes. Given a suspect among these siblings, we can use the data of honest siblings to testify for or against the suspect. We can calculate the passage probability of the common path using only the honest receivers and recalculate the same in the presence of the suspect. If there is significant difference in the two estimates, we can conclude that the suspect must have cheated.

In certain applications, the logical topology of a multicast tree is available using tools such as `mtrace` [8]. Consider the case when we are given any set of honest receivers and a suspect, along with the part of logical multicast tree topology containing them. These honest receivers may or may not be the siblings of the suspect. But if the honest receivers and the suspect share a common path

from the sender to their ancestor in such a manner that passage probability of this common path can be estimated using only the honest receivers and the passage probability of the same path can be estimated along with the suspect, we can still detect if the suspect had cheated. Consider the case when we have two honest receivers (A,B) and a suspect (C), and suppose we know that they connect together as shown in Fig 6(b). We can calculate the passage probability of the path SD using only A and B as P_{ab}. If this changes when we recalculate it along with C, we can conclude that C is cheating. However, consider the case when A, B and C connect together as shown in figure 6(a). Here, using the honest receivers we can calculate the passage probability of the path SE as P_{ab}. When C cheats the passage probabilities of path SD and DE both change, but their product still remains equal to P_{ab}. In this example, there is no any common path whose passage probability can be calculated both with and without the suspect C. Thus, even knowing the topology does not help if the honest receivers are not located in the right positions in the multicast tree where detection must be performed. In fact, if we install exactly one honest receiver at the right position, we can perform detection without use of topology information.

Algorithm `DetectMisbehaviour`

A : *Honest receiver at correct position*
B : *Any honest receiver*
C : *Suspect*
α : *Threshold*

1. $P_a^b \leftarrow (n_{11*})/(n_{*1*})$
   $P_a^c \leftarrow (n_{1*1})/(n_{1**})$
   $P_{ab} \leftarrow (n_{1**} \cdot n_{*1*})/(n_{11*} \cdot N)$
   $P_{ac} \leftarrow (n_{1**} \cdot n_{**1})/(n_{1*1} \cdot N)$

2. `If` $|P_a^b - P_a^c| > \alpha$ `or` $|P_{ab} - P_{ac}| > \alpha$
   `print` *Cheating*
   `else`
   `print` *No Cheating*

Figure 7: Algorithm to detect misbehavior with two honest receivers and a suspect

5.3.1 Detection with two honest receivers

For detection, we advocate the installation of one honest receiver in such a manner that all receivers in the multicast tree share a common path with this honest receiver. This is shown in figure 6(d). Receiver A joins that router D closest to S, to which any other receiver can join directly, guaranteeing the existence of the common path SD. Receiver B can be placed anywhere in the multicast tree. This ensures that when we are given two honest receivers and a suspect, they always connect together using topology shown in figure 6(b) or (c). The condition for detection is same for both these topologies and is shown in the following algorithm `DetectMisbehaviour`. When C cheats, it

changes P_a, P_{abc} and P_b. The algorithm checks for changes in P_a and P_{abc} since P_b cannot be checked for in case of topology 6(c). If we are unable to place an honest receiver A as described,

Passage Probability	Topology Figure6(b)	Topology Figure6(c)
P_d	0.8	0.8
P_d^{minc}	0.786	0.805
P_d^{cheat}	0.875 $\Uparrow$	0.890 $\Uparrow$
P_a	0.9	0.9
P_a^{minc}	0.911	0.894
P_a^{cheat}	0.818 $\Downarrow$	0.809 $\Downarrow$
P_b	0.85	0.85
P_b^{minc}	0.861	0.887
P_b^{cheat}	0.774 $\Downarrow$	0.712 $\Downarrow$
P_c	0.6	0.7
P_c^{minc}	0.610	0.711
P_c^{cheat}	0.777 $\Uparrow$	0.825 $\Uparrow$
P_e		0.0.8
P_e^{minc}		0.785
P_e^{cheat}		0.885 $\Uparrow$

Figure 8: Results of 1000 probes and 20% cheating

then our detection procedure can be used only in certain circumstances. If the topology information is given and if the suspect forms topologies (b) or (c), we can detect misbehavior otherwise not. If the topology information is not given, we can infer topology based on passage probabilities. But these passage probabilities change due to cheating. Thus if the original topology was (a) we will still infer (a) and we cannot detect. If the topology was (b) we will infer (a) and may not detect. However if the original topology was (c) and C did not cheat much we may still infer (c) or (b). Thus whenever we infer (C) or (b), we can still use the misbehavior detection algorithm described above. The complexity our algorithm to detect misbehavior is extremely low. Note that we do not use the detailed MINC inference engine.

5.3.2 Examples of Detection

We have conducted several simulations to verify our detection algorithm. Here we present a sample example of detection. We used simple C programs to create random losses for the topologies in figure 6(b) and (c) and used our algorithm to detect misbehavior. Figure 8 shows the original passage probabilities, the passage probabilities calculated by MINC without cheating and with receiver C cheating for 20% of the probes. The results shown are for N = 1000 probes. Essentially, receiver C cheats using $MM(0 \leftarrow 1, N, 0.2)$. Figures 9, 10, 11, 12 show the differences in P_a and P_d calculated by our algorithm to check misbehavior when the amount of cheating varies from 2.5% to 40%. When we have large probe sets (>= 1000), we can afford to keep α low. However, when probe

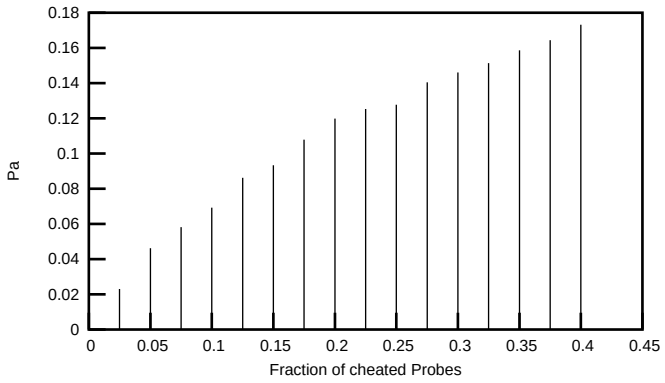

Figure 9: Differences in P_a for topology figure 6(b)

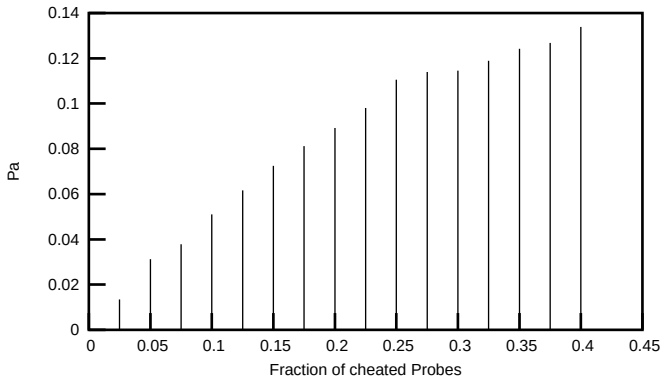

Figure 10: Differences in P_a for topology figure 6(c)

sets are small, the differences in P_a and P_d when receivers are not cheating is not negligible and we need to keep α large. Also, when we have high passage probabilities cheating by C does not cause too much increase in passage probabilities and hence differences remain low and become stagnant when P_d reaches 1. In our examples we observed that for probes larger than 1000, maintaining $\alpha = 3\%$ detects misbehaviour reliably.

5.4 Prevention Overview

In order to prevent receivers from cheating while reporting MINC feedback, we present a simple mechanism based on per packet nonces. The sender adds a random r bit nonce to every probe packet. When a receiver periodically reports its feedback for a group of probes, other than per packet binary feedback, it is obliged to send an XOR of the nonces for the packets it has received.

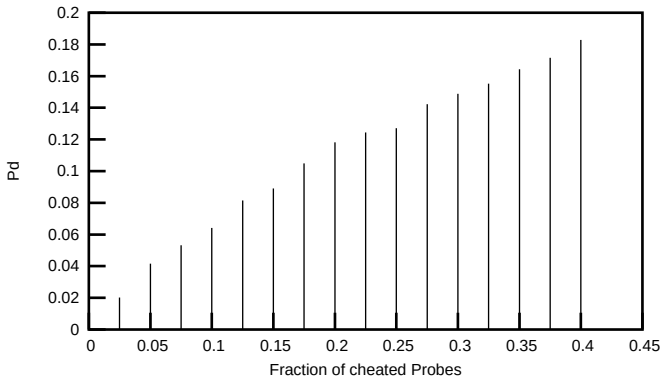

Figure 11: Differences in P_d for topology figure 6(b)

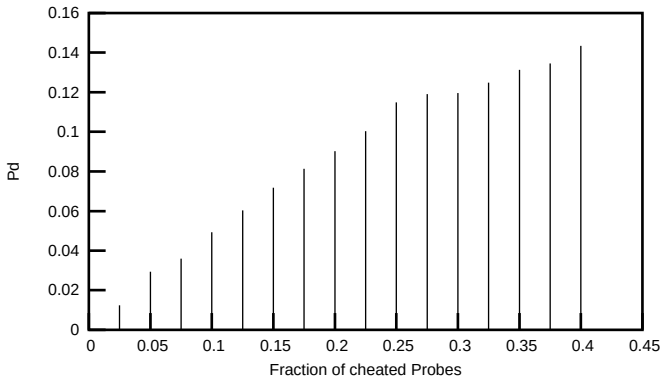

Figure 12: Differences in P_d for topology figure 6(c)

In this manner a receiver is able to give a proof of what it has received. If a receiver still wishes to cheat by reporting a 1 instead of 0 for it must guess the proof correctly. Thus it succeeds to cheat with a low probability of $1/2^r$, but is caught with a high probability of $(1 - 1/2^r)$.

Another technique to prevent receivers from cheating is for the sender to randomly skip certain sequence numbers to be used in probe packets. A receiver which did not receive a probe packet with a certain sequence number is not sure whether this packet was lost or was not sent by the sender. If it cheats and reports a 1 for a skipped packet, it is caught. However this mechanism may interfere with certain applications which use MINC. When probe packets are just the data packets sent on the multicast tree, this may interfere with the congestion control mechanism at the receiver which will interpret a missing sequence number as a loss. And if the sender, to keep interference minimum, skips very few sequence numbers, a receiver can cheat without getting caught with high probability.

6 Misbehavior Prevention in RDCC

In RDCC, the sender encodes data into several layers each of which is made available on a different multicast group. Each RDCC protocol lays down certain rules for congestion control which the receivers must follow. Broadly, these rules ask the receiver to subscribe to layers (multicast groups) when there is no congestion and unsubscribe to them when congestion occurs. However, each RDCC protocol defines congestion differently. In FLID-DL, receivers must reduce their subscription level when they experience a packet loss. In RLM, depending on the state in which the receiver is, it must reduce its subscription level based on either a single packet loss or when the number of losses exceed a certain threshold. In WEBRC, each receiver uses TFRC like equations to calulate a target rate based on loss probability and multicast round trip time. It can subscribe to a wave channel (layer) if after joining, the aggregate rates of all its channels does not exceed the target rate. Each wave channel has an exponentially decreasing rate and thus if a receiver does not join wave channels, its reception rate decreases. Receivers leave a wave channel when it becomes quiescent.

A receiver who cheats, disobeys the rules of congestion control and inflates its subsciption level to cause congestion. To solve this problem, we would like to have a prevention mechanism which ensures that each receiver behaves according to the rules laid down by its congestion control algorithm. In this context, DELTA(Distribution of ELigibility to Access) was recently proposed [10]. DELTA is a general encoding scheme for in-band distribution of keys which guard the access to multicast groups. A receiver can subscribe to a multicast group only if it has the key for that group. The key in encoded within the data packets in such a manner that if there is congestion, a receiver does not receive the key and hence cannot subscribe to the multicast group. Since different protocols define congestion differently, DELTA is applied (instantiated) differently for different protocols (DELTA is protocol specific function of congestion [10]). For example in FLID-DL, a key must be encoded within the data packets in such a manner that when a single packet is lost, the key must be lost. In addition, DELTA must take into consideration the rules of access to multicast groups as specified by the protocol. For example, in FLID-DL, if a receiver which has subscribed to groups $0..i$ experiences congestion (looses a packet), it must loose the key for group i, but receive keys for groups $0..i-1$ so that it can continue its subscription to these groups. Thus for each protocol, ideally, DELTA must respect both the definition of congestion and the rules of access to multicast groups.

In general, to implement DELTA, we want to divide a key into n components such that atleast k components are required to reconstruct the key. This is the problem of threshold secret sharing or error control coding. In (k, n) threshold secret sharing, we want to divide a secret among n components such that any k components can be used to reconstruct the secret and knowledge of any $k-1$ components does not reveal any information about the secret. Similarly, in error control coding, we want to transmit information using n symbols in such a manner that if at most $n-k$ symbols are lost (in error), we can reconstruct the information, otherwise not. The solution to the threshold secret sharing problem was proposed by Adi Shamir in [18]. It was later remarked in [13] that shamir's scheme was a particular form of Reed-Solomon coding. Shamir's threshold secret sharing scheme is based on polynomial interpolation. Given k points in two dimensional plane $(x_1, y_1)...(x_k, y_k)$, with distinct x_i's there is a unique polynomial $q(x)$ of degree $\leq k-1$, such that $q(x_i) = y_i$ for all i. Thus to share a secret S, a random polynomial of degree exactly

$k-1$ with non-zero coefficients, $q(x) = a_0 + a_1x_1 + ...a_{k-1}x_{k-1}$ is choosen with $a_0 = S$. Each component S_i is the tuple $(x_i, y_i) = (i, q(i))$, $1 \leq i \leq n$. Any set of k components can be used to compute the polynomial $q(x)$ using lagarange interpolation formula. If the k available components are $(x_1, y_1)...(x_k, y_k)$, then $q(x)$ is given by

$$\begin{aligned} q(x) &= \frac{(x-x_2)(x-x_3)...(x-x_k)}{(x_1-x_2)(x_1-x_3)...(x_1-x_k)}y_1 \\ &+ \frac{(x-x_1)(x-x_2)...(x-x_k)}{(x_2-x_1)(x_2-x_3)...(x_2-x_k)}y_2 + \dots \\ &+ \frac{(x-x_1)(x-x_2)...(x-x_{k-1})}{(x_k-x_1)(x_k-x_2)...(x_k-x_{k-1})}y_k \end{aligned}$$

The Secret S is got back by computing $q(0)$. Any set of $k-1$ or fewer components gives us no knowledge of the original $q(x)$ used. A nice property of this scheme is that secret decoding can be done without knowing k, by just consecutively decoding the secret for increasing values of k until two consecutive secrets are same.

In general, with DELTA, the time at sender is divided into epochs. During each epoch i, the sender encodes the key within data packets which the receiver must use to subscribe to the multicast group during epoch $i+2$. Thus the receiver has enough time to decode and request subscription. For cumulative layered multicast, each receiver subscribes to a stack of layers and reduces its subscription level by 1 during congestion. Thus to encode the key for any layer i, all data packets in layers $0..i$ must be used. Hence the components of key for layer i are places in all data packets from layer $0..i$. When a receiver which has subscribed to layers $0..i$ experiences congestion, it looses the key for layer i and cannot continue its subscription to layer i. Now, consider a sender with maximum of L layers. The data packets of base layer 0 must hold key components of layers $0..L$. Data packets of layer 1 must hold key components of layers $1..L$ and so on. Thus there is significant overhead. This overhead can be reduced if keys can reuse components, that is the key for layer i can be formed using key for layer $i-1$ and some additional components (which are placed in data packets of layer i). This is not possible if we use shamir's threshold scheme directly. A scheme which reuses key components was proposed in [10]. But this can only be used for protocols which define congestion as a single packet loss(FLID-DL).

Each RDCC protocol defines congestion differently and has certain rules to access multicast groups. Denote by $RC(p)$, the definition of congestion in protocol p. Denote by $RA(p)$ the rules of access to multicast groups for protocol p. In RLM, depending on the state in which the receiver is, $RC(RLM)$ is defined as a single packet loss or when losses exceed a threshold. $RC(WEBRC)$ advocates maintaining receiver reception rate less than a calculated target rate based on TFRC like equations. DELTA for a protocol p tries to design keys which ensure that the receiver respects both $RC(p)$ and $RA(p)$. For most protocols, it is possible to design keys which respect RA. But for protocols such as RLM and WEBRC, it is very difficult to design keys which can ensure that the receiver behaves exactly as specified by its RC. To tackle this problem, observe that irrespective of which protocol a receiver uses, when it inflates it subscription, it experiences congestion. During

this congestion period, it encounters closely spaced losses and bursts. Thus, irrespective of which protocol p the receiver follows, if we can provide a light protection mechanism against general congestion, we can prevent receivers from inflating their subscription. Such a mechanism can maintain a high congestion bound so that it does not interfere with $RC(p)$ but traps excessive congestion. In subsequent section we show the design of a key which just traps congestion irrespective of the protocol. We also show that such a key can be used with low overhead to ensure $RA(p)$ for cumulative layered protocols since reuse of key components is possible.

6.1 Key Design to Trap Congestion

First, we show the design for a single layer. Consider the set of packets transmitted during an epoch of time. We divide the consecutive packets into blocks of size $2S$. Each block of $2S$ packets share a small secret which in placed in them using shamir's $(S, 2S)$ threshold secret sharing. In addition, certain packets of two consecutive blocks share a secret. Packets $S..2S$ of block i and packets $1..S$ of block $i + 1$ share a secret using $(S, 2S)$ threshold secret sharing (Figure 13). The key for a layer is the sum of secrets within the blocks.

$$K = \Sigma_{i=1}^{i \leq n} k_i \tag{10}$$

When any S consecutive packets are lost, the secret is lost. Also, within any group of 2S consecutive packets if S are lost, the secret is lost. Thus in general, the secret is lost when losses are closely spaced. When losses are quite far off from each other, the secret is not lost. By adjusting parameter S, we can apply this key design for any protocol. For cumulative layered protocols, the key for layer l can be defined as the sum of keys for layers $0..l$. Thus, key components can be reused for low overhead per packet.

$$K_l = \Sigma_{j=0}^{j \leq l} \Sigma_{i=1}^{i \leq n} k_{il} \tag{11}$$

Note that, we can use any one way function instead of sum such as sum modulo , XOR, etc. Also, as in [10], the sender can precompute keys without knowing the number of packets to be transmitted. The advantage of our key design is that it can be used with most protocols. It is flexible

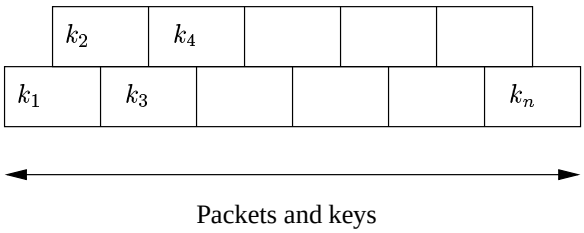

Figure 13: Key design to trap congestion

and can trap misbehavior due to inflated subscriptions. But since we are designing keys independent of specific congestion definition, any scheme which is protocol specific is definitely more effective.

7 Conclusions

In this report we showed that the loss rates of links calculated by MINC can help to detect misbehavior in SDCC protocols. For this, we require that MINC itself be immune to misbehavior. We showed how we can detect and prevent misbehavior within MINC. As future work we would like to explore how we can actually incorporate MINC inference in various source driven multicast congestion control protocols. We would also like to study if misbehavior can be detected if certain feedback reports are lost. For receiver driven congestion control protocols, we showed how to design keys which can be applied to most RDCC protocols, by keeping key design independent of specific definitions of congestion. We would like to investigate the use of error control coding in key design, in particular the use of burst error correcting codes.

8 Acknowledgments

We thank Timur Friedman for various fruitful discussions. We also want to thank Tian Bu and Wei Wei for the MINC inference engine `eminfer` which we used extensively for testing. We would also like to thank Chadi Barakat and Sara Alouf for helpful discussions.

References

[1] A. Adams, T. Bu, T. Caceres, N.G.Duffield, T. Friedman, J.Horowitz, F.Lo Presti, S.B. Moon, V. Paxson, and D. Towsley. The use of end-to-end multicast measurements for characterising internal network behavior. *IEEE Communications Magazine*, May 2000.

[2] Célio Albuquerque, Brett J.Vickers, and T.Suda. Source-Adaptive Multi-Layered Multicast Algorithms. *IEEE/ACM transactions on Networking*, Dec 2000.

[3] J. Byers, M. Frumin, G. Horn, M.Luby, and M.Mitzenmacher. FLID-DL: Congestion control for layered multicast. *NGC*, 2000.

[4] R. Caceres, N. G. Duffield, J. Horowitz, and D. Towsley. Multicast-based inference of network-internal loss characteristics. *IEEE Transactions on Information Theory*, 7:2462–2480, Nov 1999.

[5] R. Caceres, N.G. Duffield, and T. Friedman. Impromptu measurement infrastructures using RTP. *IEEE Infocom 2002*, June 2002.

[6] S.Y. Cheung, M.H. Ammar, and X. Li. On the use of destination set grouping to improve fairness in multicast video distribution. *IEEE Infocom*, March 1996.

[7] N. Duffield, J. Horowitz, F. LoPresti, and D. Towsley. Multicast topology inference from measured end-to-end loss. *IEEE Transactions on Information Theory*, 48(1):26–45, 2002.

[8] W. Fenner and S. Casner. A traceroute facility for IP multicast. *draft-ietf-idmr-traceroute-ipm-06.txt*, 2000.

[9] S. Floyd. Congestion control principles. *RFC 2914*, Sept 2000.

[10] S. Gorinsky, S. Jain, H. Vin, and Y. Zhang. Robustness to inflated subscription in multicast congestion control. *ACM SIGCOMM*, August 2003 (To appear).

[11] Sergey Gorinsky, Sugat Jain, and Harrick Vin. Multicast congestion control with distrusted receivers. *NGC*, 2002.

[12] Steven McCanne, Van Jacobson, and Martin Vetterli. Receiver-driven layered multicast. *ACM SIGCOMM*, pages 117–130, Aug 1996.

[13] R. J. McEliece and D. Sarwate. On sharing secrets and reed-solomon codes. *Communications of the ACM*, 24:583–584, 1981.

[14] S. Paul, K.Sabnani, J.C Lin, and S. Bhattacharya. Reliable multicast transport protocol(RMTP). *IEEE Journal on selected areas in Communication*, April 1997.

[15] Sylvia Ratnasamy and Steven McCanne. Inference of multicast routing trees and bottleneck bandwidths using end-to-end measurements. *IEEE INFOCOM*, 1999.

[16] L. Rizzo. A TCP-friendly single rate multicast congestion control scheme. *ACM SIGCOMM*, 2000.

[17] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. *RFC 1889*, 1996.

[18] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[19] D Sisalem and A wolisz. MLDA: A TCP-friendly congestion control framework for heterogeneous multicast environments. *IWQos*, June 2000.

[20] V. Vicisano, L. Rizzo, and J Crowcroft. TCP-like congestion control for layered multicast data transfer. *IEEE INFOCOM*, 98.

[21] J. Vieron, T. Turletti, K. Salamatian, and C. Guillemot. Source and channel adaptive rate control for multicast layered video transmission based on a clustering algorithm. *EURASIP, Special Issue on Multimedia over IP and Wireless Networks*, 2004 (To appear).

[22] J. Widmer and M. Handley. Extending equation-based congestion control to multicast applications. *ACM SIGCOMM*, 2001.

Contents

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifi que
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399