
120777 Wed Apr 3 12:38:54 2013
 new/usr/src/uts/common/io/mac/mac_sched.c
 918 Need better IP fanout (esp. with VLANs present)

```

1  /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2010 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */
25 /*
26 * Copyright 2011 Joyent, Inc. All rights reserved.
27 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
28 */

28 #include <sys/types.h>
29 #include <sys/callb.h>
30 #include <sys/sdt.h>
31 #include <sys/strsubr.h>
32 #include <sys/strsun.h>
33 #include <sys/vlan.h>
34 #include <sys/stack.h>
35 #include <sys/archsystem.h>
36 #include <inet/ipsec_impl.h>
37 #include <inet/ip_impl.h>
38 #include <inet/sadb.h>
39 #include <inet/ipsecesp.h>
40 #include <inet/ipsec.h>
41 #include <inet/ip6.h>

43 #include <sys/mac_impl.h>
44 #include <sys/mac_client_impl.h>
45 #include <sys/mac_client_priv.h>
46 #include <sys/mac_soft_ring.h>
47 #include <sys/mac_flow_impl.h>

49 static mac_tx_cookie_t mac_tx_single_ring_mode(mac_soft_ring_set_t *, mblk_t *,
50     uintptr_t, uint16_t, mblk_t **);
51 static mac_tx_cookie_t mac_tx_serializer_mode(mac_soft_ring_set_t *, mblk_t *,
52     uintptr_t, uint16_t, mblk_t **);
53 static mac_tx_cookie_t mac_tx_fanout_mode(mac_soft_ring_set_t *, mblk_t *,
54     uintptr_t, uint16_t, mblk_t **);
55 static mac_tx_cookie_t mac_tx_bw_mode(mac_soft_ring_set_t *, mblk_t *,
56     uintptr_t, uint16_t, mblk_t **);
57 static mac_tx_cookie_t mac_tx_aggr_mode(mac_soft_ring_set_t *, mblk_t *,
58     uintptr_t, uint16_t, mblk_t **);

```

```

60 typedef struct mac_tx_mode_s {
61     mac_tx_srs_mode_t      mac_tx_mode;
62     mac_tx_func_t          mac_tx_func;
63 } mac_tx_mode_t;
64
65 unchanged_portion_omitted
66
530 /*
531 * In general we do port based hashing to spread traffic over different
532 * soft rings. The below tunables allow to override that behavior. Setting one
533 * (depending on IPv6 or IPv4) to B_TRUE allows a fanout based on src
534 * IPv6 or IPv4 address. This behavior is also applicable to IPv6 packets
535 * carrying multiple optional headers and other uncommon packet types.
536 * soft rings. The below tunable allows to override that behavior. Setting it
537 * to B_TRUE allows to do a fanout based on src ipv6 address. This behavior
538 * is also the applicable to ipv6 packets carrying multiple optional headers
539 * and other uncommon packet types.
540 */
541 boolean_t mac_src_ipv6_fanout = B_FALSE;
542 boolean_t mac_src_ipv4_fanout = B_FALSE;
543
544 /*
545 * Pair of local and remote ports in the transport header
546 */
547 #define PORTS_SIZE 4
548
549 /*
550 * mac_rx_srs_proto_fanout
551 * This routine delivers packets destined to an SRS into one of the
552 * protocol soft rings.
553 *
554 * Given a chain of packets we need to split it up into multiple sub chains
555 * destined into TCP, UDP or OTH soft ring. Instead of entering
556 * the soft ring one packet at a time, we want to enter it in the form of a
557 * chain otherwise we get this start/stop behaviour where the worker thread
558 * goes to sleep and then next packets comes in forcing it to wake up etc.
559 */
560 static void
561 mac_rx_srs_proto_fanout(mac_soft_ring_set_t *mac_srs, mblk_t *head)
562 {
563     struct ether_header *ehp;
564     struct ether_vlan_header *evhp;
565     uint32_t sap;
566     ipha_t *ipha;
567     uint8_t *dstaddr;
568     size_t hdrsize;
569     mblk_t *mp;
570     mblk_t *headmp[MAX_SR_TYPES];
571     mblk_t *tailmp[MAX_SR_TYPES];
572     int cnt[MAX_SR_TYPES];
573     size_t sz[MAX_SR_TYPES];
574     size_t sz1;
575     boolean_t bw_ctl;
576     boolean_t hw_classified;
577     boolean_t dls_bypass;
578     boolean_t is_ether;
579     boolean_t is_unicast;
580     enum pkt_type type;
581     mac_client_impl_t *mcip = mac_srs->srs_mcip;
582
583     is_ether = (mcip->mci_mip->mi_info.mi_nativemedia == DL_ETHER);
584     bw_ctl = ((mac_srs->srs_type & SRST_BW_CONTROL) != 0);
585
586     /*
587      * If we don't have a Rx ring, S/W classification would have done
588      * its job and its a packet meant for us. If we were polling on

```

```

586     * the default ring (i.e. there was a ring assigned to this SRS),
587     * then we need to make sure that the mac address really belongs
588     * to us.
589     */
590     hw_classified = mac_srs->srs_ring != NULL &&
591     mac_srs->srs_ring->mr_classify_type == MAC_HW_CLASSIFIER;

593     /*
594     * Special clients (eg. VLAN, non ether, etc) need DLS
595     * processing in the Rx path. SRST_DLS_BYPASS will be clear for
596     * such SRSS. Another way of disabling bypass is to set the
597     * MCIS_RX_BYPASS_DISABLE flag.
598     */
599     dls_bypass = ((mac_srs->srs_type & SRST_DLS_BYPASS) != 0) &&
600     ((mcip->mci_state_flags & MCIS_RX_BYPASS_DISABLE) == 0);

602     bzero(headmp, MAX_SR_TYPES * sizeof (mblk_t *));
603     bzero(tailmp, MAX_SR_TYPES * sizeof (mblk_t *));
604     bzero(cnt, MAX_SR_TYPES * sizeof (int));
605     bzero(sz, MAX_SR_TYPES * sizeof (size_t));

607     /*
608     * We got a chain from SRS that we need to send to the soft rings.
609     * Since queues for TCP & IPv4 sap poll their soft rings (for
610     * performance reasons), we need to separate out v4_tcp, v4_udp
611     * and the rest goes in other.
612     */
613     while (head != NULL) {
614         mp = head;
615         head = head->b_next;
616         mp->b_next = NULL;

618         type = OTH;
619         sz1 = (mp->b_cont == NULL) ? MBLKL(mp) : msgdsize(mp);

621         if (is_ether) {
622             /*
623             * At this point we can be sure the packet at least
624             * has an ether header.
625             */
626             if (sz1 < sizeof (struct ether_header)) {
627                 mac_rx_drop_pkt(mac_srs, mp);
628                 continue;
629             }
630             ehpptr = (struct ether_header *)mp->b_rptr;

632             /*
633             * Determine if this is a VLAN or non-VLAN packet.
634             */
635             if ((sap = ntohs(ehpptr->ether_type)) == VLAN_TPID) {
636                 evhp = (struct ether_vlan_header *)mp->b_rptr;
637                 sap = ntohs(evhp->ether_type);
638                 hdrsize = sizeof (struct ether_vlan_header);
639                 /*
640                 * Check if the VID of the packet, if any,
641                 * belongs to this client.
642                 */
643                 if (!mac_client_check_flow_vid(mcip,
644                     VLAN_ID( ntohs(evhp->ether_tci) ))) {
645                     mac_rx_drop_pkt(mac_srs, mp);
646                     continue;
647                 }
648             } else {
649                 hdrsize = sizeof (struct ether_header);
650             }
651             is_unicast =

```

```

652         (((uint8_t *)&ehpptr->ether_dhost)[0] & 0x01) == 0);
653         dstaddr = (uint8_t *)&ehpptr->ether_dhost;
654     } else {
655         mac_header_info_t mhi;

657         if (mac_header_info((mac_handle_t)mcip->mci_mip,
658             mp, &mhi) != 0) {
659             mac_rx_drop_pkt(mac_srs, mp);
660             continue;
661         }
662         hdrsize = mhi.mhi_hdrsize;
663         sap = mhi.mhi_bindsap;
664         is_unicast = (mhi.mhi_dsttype == MAC_ADDRTYPE_UNICAST);
665         dstaddr = (uint8_t *)mhi.mhi_daddr;
666     }

668     if (!dls_bypass) {
669         FANOUT_ENQUEUE_MP(headmp[type], tailmp[type],
670             cnt[type], bw_ctl, sz[type], sz1, mp);
671         continue;
672     }

674     if (sap == ETHERTYPE_IP) {
675         /*
676         * If we are H/W classified, but we have promisc
677         * on, then we need to check for the unicast address.
678         */
679         if (hw_classified && mcip->mci_promisc_list != NULL) {
680             mac_address_t *map;

682             rw_enter(&mcip->mci_rw_lock, RW_READER);
683             map = mcip->mci_unicast;
684             if (bcmp(dstaddr, map->ma_addr,
685                 map->ma_len) == 0)
686                 type = UNDEF;
687             rw_exit(&mcip->mci_rw_lock);
688         } else if (is_unicast) {
689             type = UNDEF;
690         }
691     }

693     /*
694     * This needs to become a contract with the driver for
695     * the fast path.
696     *
697     * In the normal case the packet will have at least the L2
698     * header and the IP + Transport header in the same mblk.
699     * This is usually the case when the NIC driver sends up
700     * the packet. This is also true when the stack generates
701     * a packet that is looped back and when the stack uses the
702     * fastpath mechanism. The normal case is optimized for
703     * performance and may bypass DLS. All other cases go through
704     * the 'OTH' type path without DLS bypass.
705     */

707     ipha = (ipha_t *) (mp->b_rptr + hdrsize);
708     if ((type != OTH) && MBLK_RX_FANOUT_SLOWPATH(mp, ipha))
709         type = OTH;

711     if (type == OTH) {
712         FANOUT_ENQUEUE_MP(headmp[type], tailmp[type],
713             cnt[type], bw_ctl, sz[type], sz1, mp);
714         continue;
715     }

717     ASSERT(type == UNDEF);

```

```

718      /*
719      * We look for at least 4 bytes past the IP header to get
720      * the port information. If we get an IP fragment, we don't
721      * have the port information, and we use just the protocol
722      * information.
723      */
724      switch (ipha->ipha_protocol) {
725      case IPPROTO_TCP:
726          type = V4_TCP;
727          mp->b_rptr += hdrsize;
728          break;
729      case IPPROTO_UDP:
730          type = V4_UDP;
731          mp->b_rptr += hdrsize;
732          break;
733      default:
734          type = OTH;
735          break;
736      }

738      FANOUT_ENQUEUE_MP(headmp[type], tailmp[type], cnt[type],
739      bw_ctl, sz[type], sz1, mp);
740  }

742  for (type = V4_TCP; type < UNDEF; type++) {
743      if (headmp[type] != NULL) {
744          mac_soft_ring_t          *softring;

746          ASSERT(tailmp[type]->b_next == NULL);
747          switch (type) {
748          case V4_TCP:
749              softring = mac_srs->srs_tcp_soft_rings[0];
750              break;
751          case V4_UDP:
752              softring = mac_srs->srs_udp_soft_rings[0];
753              break;
754          case OTH:
755              softring = mac_srs->srs_oth_soft_rings[0];
756          }
757          mac_rx_soft_ring_process(mcip, softring,
758          headmp[type], tailmp[type], cnt[type], sz[type]);
759      }
760  }
761 }

763 int      fanout_unaligned = 0;
763 int      fanout_unaligned = 0;

765 /*
766 * mac_rx_srs_long_fanout
767 *
768 * The fanout routine for VLANs, and for anything else that isn't performing
769 * explicit dls bypass. Returns -1 on an error (drop the packet due to a
770 * malformed packet), 0 on success, with values written in *indx and *type.
771 * The fanout routine for IPv6
772 */
772 static int
773 mac_rx_srs_long_fanout(mac_soft_ring_set_t *mac_srs, mblk_t *mp,
774      uint32_t sap, size_t hdrsize, enum pkt_type *type, uint_t *indx)
775 {
776     ip6_t      *ip6h;
777     ipha_t      *ipha;
778     uint8_t      *wherep;
779     uint_t      hash;
780     uint16_t      remlen;
781     uint8_t      nexthdr;

```

```

782     uint16_t      hdr_len;
783     uint32_t      src_val;

781     if (sap == ETHERTYPE_IPV6) {
784         boolean_t      modifiable = B_TRUE;
785         boolean_t      v6;

787         ASSERT(MBLKL(mp) >= hdrsize);

789         if (sap == ETHERTYPE_IPV6) {
790             v6 = B_TRUE;
791             hdr_len = IPV6_HDR_LEN;
792         } else if (sap == ETHERTYPE_IP) {
793             v6 = B_FALSE;
794             hdr_len = IP_SIMPLE_HDR_LENGTH;
795         } else {
796             *indx = 0;
797             *type = OTH;
798             return (0);
799         }

801         ip6h = (ip6_t *) (mp->b_rptr + hdrsize);
802         ipha = (ipha_t *) ip6h;

804         if ((uint8_t *) ip6h == mp->b_wptr) {
787             if ((unsigned char *) ip6h == mp->b_wptr) {
805                 /*
806                  * The first mblk_t only includes the mac header.
807                  * Note that it is safe to change the mp pointer here,
808                  * as the subsequent operation does not assume mp
809                  * points to the start of the mac header.
810                  */
811                 mp = mp->b_cont;

813                 /*
814                  * Make sure the IP header points to an entire one.
815                  * Make sure ip6h holds the full ip6_t structure.
816                  */
816                 if (mp == NULL)
817                     return (-1);

819                 if (MBLKL(mp) < hdr_len) {
820                     if (MBLKL(mp) < IPV6_HDR_LEN) {
821                         modifiable = (DB_REF(mp) == 1);

822                     if (modifiable && !pullupmsg(mp, hdr_len))
823                         if (modifiable &&
824                             !pullupmsg(mp, IPV6_HDR_LEN)) {
825                             return (-1);
826                         }
827                     }
828                 }

826                 ip6h = (ip6_t *) mp->b_rptr;
827                 ipha = (ipha_t *) ip6h;
828             }

830             if (!modifiable || !(OK_32PTR((char *) ip6h)) ||
831                 ((uint8_t *) ip6h + hdr_len > mp->b_wptr)) {
832                 if ((unsigned char *) ip6h + IPV6_HDR_LEN > mp->b_wptr) {
833                     /*
834                      * If either the IP header is not aligned, or it does not hold
835                      * the complete simple structure (a pullupmsg() is not an
836                      * option since it would result in an unaligned IP header),
837                      * fanout to the default ring.
838                      * Note that this may cause packet reordering.

```

```

817      * If either ip6h is not aligned, or ip6h does not
818      * hold the complete ip6_t structure (a pullupmsg())
819      * is not an option since it would result in an
820      * unaligned ip6h), fanout to the default ring. Note
821      * that this may cause packets reordering.
822
823      */
824      *indx = 0;
825      *type = OTH;
826      fanout_unaligned++;
827      fanout_unaligned++;
828      return (0);
829  }
830
831  /*
832   * Extract next-header, full header length, and source-hash value
833   * using v4/v6 specific fields.
834   */
835  if (v6) {
836      remlen = ntohs(ip6h->ip6_plen);
837      nexthdr = ip6h->ip6_nxt;
838      src_val = V4_PART_OF_V6(ip6h->ip6_src);
839
840      if (remlen < MIN_EHDR_LEN)
841          return (-1);
842
843      /*
844       * Do src based fanout if below tunable is set to B_TRUE or
845       * when mac_ip_hdr_length_v6() fails because of malformed
846       * packets or because mblks need to be concatenated using
847       * packets or because mblk's need to be concatenated using
848       * pullupmsg().
849       */
850      if (mac_src_ipv6_fanout || !mac_ip_hdr_length_v6(ip6h,
851          mp->b_wptr, &hdr_len, &nexthdr, NULL)) {
852          goto src_based_fanout;
853      }
854  } else {
855      hdr_len = IPH_HDR_LENGTH(ipha);
856      remlen = ntohs(ipha->ipha_length) - hdr_len;
857      nexthdr = ipha->ipha_protocol;
858      src_val = (uint32_t)ipha->ipha_src;
859      /*
860       * Catch IPv4 fragment case here.  IPv6 has nexthdr == FRAG
861       * for its equivalent case.
862       */
863      if (mac_src_ipv4_fanout ||
864          (ntohs(ipha->ipha_fragment_offset_and_flags) &
865           (IPH_MF | IPH_OFFSET)) != 0) {
866          goto src_based_fanout;
867      }
868  }
869  if (remlen < MIN_EHDR_LEN)
870      return (-1);
871  whereptr = (uint8_t *)ip6h + hdr_len;
872
873  /* If the transport is one of below, we do port/SPI based fanout */
874  /* If the transport is one of below, we do port based fanout */
875  switch (nexthdr) {
876  case IPPROTO_TCP:
877  case IPPROTO_UDP:
878  case IPPROTO_SCTP:
879  case IPPROTO_ESP:
880      /*
881       * If the ports or SPI in the transport header is not part of
882       * the mblk, do src_based_fanout, instead of calling
883       * pullupmsg().

```

```

893      */
894      if (mp->b_cont == NULL || whereptr + PORTS_SIZE <= mp->b_wptr)
895          break; /* out of switch... */
896      /* FALLTHRU */
897  default:
898      if (mp->b_cont != NULL &&
899          whereptr + PORTS_SIZE > mp->b_wptr) {
900          goto src_based_fanout;
901      }
902      break;
903  default:
904      break;
905  }
906
907  switch (nexthdr) {
908  case IPPROTO_TCP:
909      hash = HASH_ADDR(src_val, *(uint32_t *)whereptr);
910      *indx = COMPUTE_INDEX(hash, mac_srs->srs_tcp_ring_count);
911      hash = HASH_ADDR(V4_PART_OF_V6(ip6h->ip6_src),
912          *(uint32_t *)whereptr);
913      *indx = COMPUTE_INDEX(hash,
914          mac_srs->srs_tcp_ring_count);
915      *type = OTH;
916      break;
917
918  case IPPROTO_UDP:
919  case IPPROTO_SCTP:
920  case IPPROTO_ESP:
921      if (mac_fanout_type == MAC_FANOUT_DEFAULT) {
922          hash = HASH_ADDR(src_val, *(uint32_t *)whereptr);
923          hash = HASH_ADDR(V4_PART_OF_V6(ip6h->ip6_src),
924              *(uint32_t *)whereptr);
925          *indx = COMPUTE_INDEX(hash,
926              mac_srs->srs_udp_ring_count);
927      } else {
928          *indx = mac_srs->srs_ind % mac_srs->srs_udp_ring_count;
929          *indx = mac_srs->srs_ind %
930              mac_srs->srs_udp_ring_count;
931          mac_srs->srs_ind++;
932      }
933      *type = OTH;
934      break;
935
936      /* For all other protocol, do source based fanout */
937  default:
938      goto src_based_fanout;
939  }
940  } else {
941      *indx = 0;
942      *type = OTH;
943  }
944  return (0);
945
946  src_based_fanout:
947      hash = HASH_ADDR(src_val, (uint32_t)0);
948      hash = HASH_ADDR(V4_PART_OF_V6(ip6h->ip6_src), (uint32_t)0);
949      *indx = COMPUTE_INDEX(hash, mac_srs->srs_oth_ring_count);
950      *type = OTH;
951      return (0);
952  }
953  }
954  }
955  }
956  }
957  }
958  }
959  }
960  }
961  }
962  }
963  }
964  }
965  }
966  }
967  }
968  }
969  }
970  }
971  }
972  }
973  }
974  }
975  }
976  }
977  }
978  }
979  }
980  }
981  }
982  }
983  }
984  }
985  }
986  }
987  }
988  }
989  }
990  }
991  }
992  }
993  }
994  }
995  }
996  }
997  }
998  }
999  }
1000  }
1001  }
1002  }
1003  }
1004  }
1005  }
1006  }
1007  }
1008  }
1009  }
1010  }
1011  }
1012  }
1013  }
1014  }
1015  }
1016  }
1017  }
1018  }
1019  }
1020  }
1021  }
1022  }
1023  }
1024  }
1025  }
1026  }
1027  }
1028  }
1029  }
1030  }
1031  }
1032  }
1033  }
1034  }
1035  }
1036  }
1037  }
1038  }
1039  }
1040  }
1041  }
1042  }
1043  }
1044  }
1045  }
1046  }
1047  }
1048  }
1049  }
1050  }
1051  }
1052  }
1053  }
1054  }
1055  }
1056  }
1057  }
1058  }
1059  }
1060  }
1061  }
1062  }
1063  }
1064  }
1065  }
1066  }
1067  }
1068  }
1069  }
1070  }
1071  }
1072  }
1073  }
1074  }
1075  }
1076  }
1077  }
1078  }
1079  }
1080  }
1081  }
1082  }
1083  }
1084  }
1085  }
1086  }
1087  }
1088  }
1089  }
1090  }
1091  }
1092  }
1093  }
1094  }
1095  }
1096  }
1097  }
1098  }
1099  }
1100  }
1101  }
1102  }
1103  }
1104  }
1105  }
1106  }
1107  }
1108  }
1109  }
1110  }
1111  }
1112  }
1113  }
1114  }
1115  }
1116  }
1117  }
1118  }
1119  }
1120  }
1121  }
1122  }
1123  }
1124  }
1125  }
1126  }
1127  }
1128  }
1129  }
1130  }
1131  }
1132  }
1133  }
1134  }
1135  }
1136  }
1137  }
1138  }
1139  }
1140  }
1141  }
1142  }
1143  }
1144  }
1145  }
1146  }
1147  }
1148  }
1149  }
1150  }
1151  }
1152  }
1153  }
1154  }
1155  }
1156  }
1157  }
1158  }
1159  }
1160  }
1161  }
1162  }
1163  }
1164  }
1165  }
1166  }
1167  }
1168  }
1169  }
1170  }
1171  }
1172  }
1173  }
1174  }
1175  }
1176  }
1177  }
1178  }
1179  }
1180  }
1181  }
1182  }
1183  }
1184  }
1185  }
1186  }
1187  }
1188  }
1189  }
1190  }
1191  }
1192  }
1193  }
1194  }
1195  }
1196  }
1197  }
1198  }
1199  }
1200  }
1201  }
1202  }
1203  }
1204  }
1205  }
1206  }
1207  }
1208  }
1209  }
1210  }
1211  }
1212  }
1213  }
1214  }
1215  }
1216  }
1217  }
1218  }
1219  }
1220  }
1221  }
1222  }
1223  }
1224  }
1225  }
1226  }
1227  }
1228  }
1229  }
1230  }
1231  }
1232  }
1233  }
1234  }
1235  }
1236  }
1237  }
1238  }
1239  }
1240  }
1241  }
1242  }
1243  }
1244  }
1245  }
1246  }
1247  }
1248  }
1249  }
1250  }
1251  }
1252  }
1253  }
1254  }
1255  }
1256  }
1257  }
1258  }
1259  }
1260  }
1261  }
1262  }
1263  }
1264  }
1265  }
1266  }
1267  }
1268  }
1269  }
1270  }
1271  }
1272  }
1273  }
1274  }
1275  }
1276  }
1277  }
1278  }
1279  }
1280  }
1281  }
1282  }
1283  }
1284  }
1285  }
1286  }
1287  }
1288  }
1289  }
1290  }
1291  }
1292  }
1293  }
1294  }
1295  }
1296  }
1297  }
1298  }
1299  }
1300  }
1301  }
1302  }
1303  }
1304  }
1305  }
1306  }
1307  }
1308  }
1309  }
1310  }
1311  }
1312  }
1313  }
1314  }
1315  }
1316  }
1317  }
1318  }
1319  }
1320  }
1321  }
1322  }
1323  }
1324  }
1325  }
1326  }
1327  }
1328  }
1329  }
1330  }
1331  }
1332  }
1333  }
1334  }
1335  }
1336  }
1337  }
1338  }
1339  }
1340  }
1341  }
1342  }
1343  }
1344  }
1345  }
1346  }
1347  }
1348  }
1349  }
1350  }
1351  }
1352  }
1353  }
1354  }
1355  }
1356  }
1357  }
1358  }
1359  }
1360  }
1361  }
1362  }
1363  }
1364  }
1365  }
1366  }
1367  }
1368  }
1369  }
1370  }
1371  }
1372  }
1373  }
1374  }
1375  }
1376  }
1377  }
1378  }
1379  }
1380  }
1381  }
1382  }
1383  }
1384  }
1385  }
1386  }
1387  }
1388  }
1389  }
1390  }
1391  }
1392  }
1393  }
1394  }
1395  }
1396  }
1397  }
1398  }
1399  }
1400  }
1401  }
1402  }
1403  }
1404  }
1405  }
1406  }
1407  }
1408  }
1409  }
1410  }
1411  }
1412  }
1413  }
1414  }
1415  }
1416  }
1417  }
1418  }
1419  }
1420  }
1421  }
1422  }
1423  }
1424  }
1425  }
1426  }
1427  }
1428  }
1429  }
1430  }
1431  }
1432  }
1433  }
1434  }
1435  }
1436  }
1437  }
1438  }
1439  }
1440  }
1441  }
1442  }
1443  }
1444  }
1445  }
1446  }
1447  }
1448  }
1449  }
1450  }
1451  }
1452  }
1453  }
1454  }
1455  }
1456  }
1457  }
1458  }
1459  }
1460  }
1461  }
1462  }
1463  }
1464  }
1465  }
1466  }
1467  }
1468  }
1469  }
1470  }
1471  }
1472  }
1473  }
1474  }
1475  }
1476  }
1477  }
1478  }
1479  }
1480  }
1481  }
1482  }
1483  }
1484  }
1485  }
1486  }
1487  }
1488  }
1489  }
1490  }
1491  }
1492  }
1493  }
1494  }
1495  }
1496  }
1497  }
1498  }
1499  }
1500  }
1501  }
1502  }
1503  }
1504  }
1505  }
1506  }
1507  }
1508  }
1509  }
1510  }
1511  }
1512  }
1513  }
1514  }
1515  }
1516  }
1517  }
1518  }
1519  }
1520  }
1521  }
1522  }
1523  }
1524  }
1525  }
1526  }
1527  }
1528  }
1529  }
1530  }
1531  }
1532  }
1533  }
1534  }
1535  }
1536  }
1537  }
1538  }
1539  }
1540  }
1541  }
1542  }
1543  }
1544  }
1545  }
1546  }
1547  }
1548  }
1549  }
1550  }
1551  }
1552  }
1553  }
1554  }
1555  }
1556  }
1557  }
1558  }
1559  }
1560  }
1561  }
1562  }
1563  }
1564  }
1565  }
1566  }
1567  }
1568  }
1569  }
1570  }
1571  }
1572  }
1573  }
1574  }
1575  }
1576  }
1577  }
1578  }
1579  }
1580  }
1581  }
1582  }
1583  }
1584  }
1585  }
1586  }
1587  }
1588  }
1589  }
1590  }
1591  }
1592  }
1593  }
1594  }
1595  }
1596  }
1597  }
1598  }
1599  }
1600  }
1601  }
1602  }
1603  }
1604  }
1605  }
1606  }
1607  }
1608  }
1609  }
1610  }
1611  }
1612  }
1613  }
1614  }
1615  }
1616  }
1617  }
1618  }
1619  }
1620  }
1621  }
1622  }
1623  }
1624  }
1625  }
1626  }
1627  }
1628  }
1629  }
1630  }
1631  }
1632  }
1633  }
1634  }
1635  }
1636  }
1637  }
1638  }
1639  }
1640  }
1641  }
1642  }
1643  }
1644  }
1645  }
1646  }
1647  }
1648  }
1649  }
1650  }
1651  }
1652  }
1653  }
1654  }
1655  }
1656  }
1657  }
1658  }
1659  }
1660  }
1661  }
1662  }
1663  }
1664  }
1665  }
1666  }
1667  }
1668  }
1669  }
1670  }
1671  }
1672  }
1673  }
1674  }
1675  }
1676  }
1677  }
1678  }
1679  }
1680  }
1681  }
1682  }
1683  }
1684  }
1685  }
1686  }
1687  }
1688  }
1689  }
1690  }
1691  }
1692  }
1693  }
1694  }
1695  }
1696  }
1697  }
1698  }
1699  }
1700  }
1701  }
1702  }
1703  }
1704  }
1705  }
1706  }
1707  }
1708  }
1709  }
1710  }
1711  }
1712  }
1713  }
1714  }
1715  }
1716  }
1717  }
1718  }
1719  }
1720  }
1721  }
1722  }
1723  }
1724  }
1725  }
1726  }
1727  }
1728  }
1729  }
1730  }
1731  }
1732  }
1733  }
1734  }
1735  }
1736  }
1737  }
1738  }
1739  }
1740  }
1741  }
1742  }
1743  }
1744  }
1745  }
1746  }
1747  }
1748  }
1749  }
1750  }
1751  }
1752  }
1753  }
1754  }
1755  }
1756  }
1757  }
1758  }
1759  }
1760  }
1761  }
1762  }
1763  }
1764  }
1765  }
1766  }
1767  }
1768  }
1769  }
1770  }
1771  }
1772  }
1773  }
1774  }
1775  }
1776  }
1777  }
1778  }
1779  }
1780  }
1781  }
1782  }
1783  }
1784  }
1785  }
1786  }
1787  }
1788  }
1789  }
1790  }
1791  }
1792  }
1793  }
1794  }
1795  }
1796  }
1797  }
1798  }
1799  }
1800  }
1801  }
1802  }
1803  }
1804  }
1805  }
1806  }
1807  }
1808  }
1809  }
1810  }
1811  }
1812  }
1813  }
1814  }
1815  }
1816  }
1817  }
1818  }
1819  }
1820  }
1821  }
1822  }
1823  }
1824  }
1825  }
1826  }
1827  }
1828  }
1829  }
1830  }
1831  }
1832  }
1833  }
1834  }
1835  }
1836  }
1837  }
1838  }
1839  }
1840  }
1841  }
1842  }
1843  }
1844  }
1845  }
1846  }
1847  }
1848  }
1849  }
1850  }
1851  }
1852  }
1853  }
1854  }
1855  }
1856  }
1857  }
1858  }
1859  }
1860  }
1861  }
1862  }
1863  }
1864  }
1865  }
1866  }
1867  }
1868  }
1869  }
1870  }
1871  }
1872  }
1873  }
1874  }
1875  }
1876  }
1877  }
1878  }
1879  }
1880  }
1881  }
1882  }
1883  }
1884  }
1885  }
1886  }
1887  }
1888  }
1889  }
1890  }
1891  }
1892  }
1893  }
1894  }
1895  }
1896  }
1897  }
1898  }
1899  }
1900  }
1901  }
1902  }
1903  }
1904  }
1905  }
1906  }
1907  }
1908  }
1909  }
1910  }
1911  }
1912  }
1913  }
1914  }
1915  }
1916  }
1917  }
1918  }
1919  }
1920  }
1921  }
1922  }
1923  }
1924  }
1925  }
1926  }
1927  }
1928  }
1929  }
1930  }
1931  }
1932  }
1933  }
1934  }
1935  }
1936  }
1937  }
1938  }
1939  }
1940  }
1941  }
1942  }
1943  }
1944  }
1945  }
1946  }
1947  }
1948  }
1949  }
1950  }
1951  }
1952  }
1953  }
1954  }
1955  }
1956  }
1957  }
1958  }
1959  }
1960  }
1961  }
1962  }
1963  }
1964  }
1965  }
1966  }
1967  }
1968  }
1969  }
1970  }
1971  }
1972  }
1973  }
1974  }
1975  }
1976  }
1977  }
1978  }
1979  }
1980  }
1981  }
1982  }
1983  }
1984  }
1985  }
1986  }
1987  }
1988  }
1989  }
1990  }
1991  }
1992  }
1993  }
1994  }
1995  }
1996  }
1997  }
1998  }
1999  }
2000  }
2001  }
2002  }
2003  }
2004  }
2005  }
2006  }
2007  }
2008  }
2009  }
2010  }
2011  }
2012  }
2013  }
2014  }
2015  }
2016  }
2017  }
2018  }
2019  }
2020  }
2021  }
2022  }
2023  }
2024  }
2025  }
2026  }
2027  }
2028  }
2029  }
2030  }
2031  }
2032  }
2033  }
2034  }
2035  }
2036  }
2037  }
2038  }
2039  }
2040  }
2041  }
2042  }
2043  }
2044  }
2045  }
2046  }
2047  }
2048  }
2049  }
2050  }
2051  }
2052  }
2053  }
2054  }
2055  }
2056  }
2057  }
2058  }
2059  }
2060  }
2061  }
2062  }
2063  }
2064  }
2065  }
2066  }
2067  }
2068  }
2069  }
2070  }
2071  }
2072  }
2073  }
2074  }
2075  }
2076  }
2077  }
2078  }
2079  }
2080  }
2081  }
2082  }
2083  }
2084  }
2085  }
2086  }
2087  }
2088  }
2089  }
2090  }
2091  }
2092  }
2093  }
2094  }
2095  }
2096  }
2097  }
2098  }
2099  }
2100  }
2101  }
2102  }
2103  }
2104  }
2105  }
2106  }
2107  }
2108  }
2109  }
2110  }
2111  }
2112  }
2113  }
2114  }
2115  }
2116  }
2117  }
2118  }
2119  }
2120  }
2121  }
2122  }
2123  }
2124  }
2125  }
2126  }
2127  }
2128  }
2129  }
2130  }
2131  }
2132  }
2133  }
2134  }
2135  }
2136  }
2137  }
2138  }
2139  }
2140  }
2141  }
2142  }
2143  }
2144  }
2145  }
2146  }
2147  }
2148  }
2149  }
2150  }
2151  }
2152  }
2153  }
2154  }
2155  }
2156  }
2157  }
2158  }
2159  }
2160  }
2161  }
2162  }
2163  }
2164  }
2165  }
2166  }
2167  }
2168  }
2169  }
2170  }
2171  }
2172  }
2173  }
2174  }
2175  }
2176  }
2177  }
2178  }
2179  }
2180  }
2181  }
2182  }
2183  }
2184  }
2185  }
2186  }
2187  }
2188  }
2189  }
2190  }
2191  }
2192  }
2193  }
2194  }
2195  }
2196  }
2197  }
2198  }
2199  }
2200  }
2201  }
2202  }
2203  }
2204  }
2205  }
2206  }
2207  }
2208  }
2209  }
2210  }
2211  }
2212  }
2213  }
2214  }
2215  }
2216  }
2217  }
2218  }
2219  }
2220  }
2221  }
2222  }
2223  }
2224  }
2225  }
2226  }
2227  }
2228  }
2229  }
2230  }
2231  }
2232  }
2233  }
2234  }
2235  }
2236  }
2237  }
2238  }
2239  }
2240  }
2241  }
2242  }
2243  }
2244  }
2245  }
2246  }
2247  }
2248  }
2249  }
2250  }
2251  }
2252  }
2253  }
2254  }
2255  }
2256  }
2257  }
2258  }
2259  }
2260  }
2261  }
2262  }
2263  }
2264  }
2265  }
2266  }
2267  }
2268  }
2269  }
2270  }
2271  }
2272  }
2273  }
2274  }
2275  }
2276  }
2277  }
2278  }
2279  }
2280  }
2281  }
2282  }
2283  }
2284  }
2285  }
2286  }
2287  }
2288  }
2289  }
2290  }
2291  }
2292  }
2293  }
2294  }
2295  }
2296  }
2297  }
2298  }
2299  }
2300  }
2301  }
2302  }
2303  }
2304  }
2305  }
2306  }
2307  }
2308  }
2309  }
2310  }
2311  }
2312  }
2313  }
2314  }
2315  }
2316  }
2317  }
2318  }
2319  }
2320  }
2321  }
2322  }
2323  }
2324  }
2325  }
2326  }
2327  }
2328  }
2329  }
2330  }
2331  }
2332  }
2333  }
2334  }
2335  }
2336  }
2337  }
2338  }
2339  }
2340  }
2341  }
2342  }
2343  }
2344  }
2345  }
2346  }
2347  }
2348  }
2349  }
2350  }
2351  }
2352  }
2353  }
2354  }
2355  }
2356  }
2357  }
2358  }
2359  }
2360  }
2361  }
2362  }
2363  }
2364  }
2365  }
2366  }
2367  }
2368  }
2369  }
2370  }
2371  }
2372  }
2373  }
2374  }
2375  }
2376  }
2377  }
2378  }
2379  }
2380  }
2381  }
2382  }
2383  }
2384  }
2385  }
2386  }
2387  }
2388  }
2389  }
2390  }
2391  }
2392  }
2393  }
2394  }
2395  }
2396  }
2397  }
2398  }
2399  }
2400  }
2401  }
2402  }
2403  }
2404  }
2405  }
2406  }
2407  }
2408  }
2409  }
2410  }
2411  }
2412  }
2413  }
2414  }
2415  }
2416  }
2417  }
2418  }
2419  }
2420  }
2421  }
2422  }
2423  }
2424  }
2425  }
2426  }
2427  }
2428  }
2429  }
2430  }
2431  }
2432  }
2433  }
2434  }
2435  }
2436  }
2437  }
2438  }
2439  }
2440  }
2441  }
2442  }
2443  }
2444  }
2445  }
2446  }
2447  }
2448  }
2449  }
2450  }
2451  }
2452  }
2453  }
2454  }
2455  }
2456  }
2457  }
2458  }
2459
```