Payless Algorithm 1- Implemention and Setting of Tmin and Tmax

笔记本： JTAG
创建时间： 2016/1/29 4:38 **更新时间：** 2016/1/29 5:50
作者： 唐继兴

Payless 论文中给出的FlowStatisticsCollectionScheduling算法如下所示。

Algorithm 1 FlowStatisticsCollectionScheduling(*Event e*)

globals: $active\_flows$ //Currently Active Flows
$schedule\_table$ //Associative table of active flows
// indexed by poll frequency
U // Utilization Statistics. Output of this algorithm

if e is *Initialization* event **then**
  $active\_flows \leftarrow \phi$, $schedule\_table \leftarrow \phi$, $U \leftarrow \phi$
end if
if e is a `PacketIn` event **then**
  $f \leftarrow \langle e.switch, e.port, \mathcal{T}_{min}, 0 \rangle$
  $schedule\_table[\mathcal{T}_{min}] \leftarrow schedule\_table[\mathcal{T}_{min}] \cup f$
else if e is timeout τ in $schedule\_table$ **then**
  for all flows $f \in schedule\_table[\tau]$ **do**
    send a `FlowStatisticsRequest` to $f.switch$
  end for
else if e is a `FlowStatisticsReply` event for flow f **then**
  $diff\_byte\_count \leftarrow e.byte\_count - f.byte\_count$
  $diff\_duration \leftarrow e.duration - f.duration$
  $checkpoint \leftarrow current\_time\_stamp$
  $U[f.port][f.switch][checkpoint] \leftarrow \langle diff\_byte\_count, diff\_duration \rangle$
  if $diff\_byte\_count < \Delta_1$ **then**
    $f.\tau \leftarrow \min(f.\tau\alpha, \mathcal{T}_{max})$
    Move f to $schedule\_table[f.\tau]$
  else if $diff\_byte\_count > \Delta_2$ **then**
    $f.\tau \leftarrow \max(f.\tau/\beta, \mathcal{T}_{min})$
    Move f to $schedule\_table[f.\tau]$
  end if
end if

在代码中的实现：
在论文给出的源代码中的net.floodlightcontoller.netmonitor中的NETMonitor.java文件中，有FlowStatisticsCollectionScheduling算法的实现。
上述的算法伪代码与被实现的代码中相对应的部分变量表：

伪代码	代码
Tmin	MIN_SCHEDULE_TIMEOUT
Tmax	MAX_SCHEDULE_TIMEOUT
active_flows	activeFlowTable (SortedSet<FlowEntry>)
schedule_table	schedule (SchedulerTable)

该工程使用FloodlightModuleContext模块配置参数，参数的值可以在配置文件./target/bin/floodlightdefault.properties中设置。可以配置的参数有：
ALGORITHM
MIN_SCHEDULE_TIMEOUT
MAX_SCHEDULE_TIMEOUT
MIN_SCHEDULE_BYTE_THRESHOLD
MAX_SCHEDULE_BYTE_THRESHOLD
SCHEDULE_TIMEOUT_DAMPING_FACTOR
SCHEDULE_TIMEOUT_AMPLIFY_FACTOR

配置实例：

```
net.floodlightcontroller.netmonitor.NETMonitor.algorithm = payless
net.floodlightcontroller.netmonitor.NETMonitor.min_schedule_timeout = 250
net.floodlightcontroller.netmonitor.NETMonitor.max_schedule_timeout = 2500
net.floodlightcontroller.netmonitor.NETMonitor.min_schedule_byte_threshold = 10000000
net.floodlightcontroller.netmonitor.NETMonitor.max_schedule_byte_threshold = 10000000
net.floodlightcontroller.netmonitor.NETMonitor.schedule_timeout_damping_factor = 1
net.floodlightcontroller.netmonitor.NETMonitor.schedule_timeout_amplify_factor = 1
```

Important Implemention:

```
    public void updateLinkUsage(IOFSwitch sw, FlowEntry removedEntry, OFMatch match,
OFMessage msg, OFType type)
    {
        long checkPointTimeStamp = System.currentTimeMillis();
        long timeOffset = 0;
        double duration = 0.0;
        long byteCount = 0;
        FlowEntry matchedFlow = null, tmpMatchFlow = null;

        synchronized (activeFlowTable)
        {
            for (Iterator<FlowEntry> it = activeFlowTable.iterator(); it.hasNext();)
            {
                FlowEntry flow = it.next();
                if (flow.equals(removedEntry))
                {
                    tmpMatchFlow = flow;
                    break;
                }
            }
```

```
        }


        checkPointTimeStamp -= timeOffset;
        if (type.equals(OFType.FLOW_REMOVED))
        {
            logger.debug("[FLOW_MOD] Checkpoint = " + checkPointTimeStamp);
        }

        if (tmpMatchFlow != null)
        {

            logger.debug("Found a Matching Flow: " + tmpMatchFlow.toString());
            matchedFlow = tmpMatchFlow;

            switch (type)
            {
                case FLOW_REMOVED:
                    OFFlowRemoved flRmMsg = (OFFlowRemoved) msg;
                    duration = flRmMsg.getDurationSeconds() + (flRmMsg.getDurationNanoseconds() /
1e9);

                    if
(flRmMsg.getReason().equals(OFFlowRemoved.OFFlowRemovedReason.OFPRR_IDLE_TIMEOUT))
                    {
                        timeOffset = (long) flRmMsg.getIdleTimeout() * 1000;
                        duration -= flRmMsg.getIdleTimeout();
                    }
                    byteCount = flRmMsg.getByteCount();
                    matchedFlow = tmpMatchFlow.clone();
                    if (ALGORITHM.equals("payless"))
                    {
                        //duration = tmpMatchFlow.getScheduleTimeout() / 1000.0;
                        duration = ((double) flRmMsg.getDurationSeconds() + (double)
flRmMsg.getDurationNanoseconds() / 1e9)
                                - tmpMatchFlow.getDuration();

                        byteCount = flRmMsg.getByteCount() - matchedFlow.getMatchedByteCount();
                        logger.debug("Flow Removed, Matched flow bytes= " +
matchedFlow.getMatchedByteCount());
                        schedule.removeFlowEntry(tmpMatchFlow.getScheduleTimeout(),
tmpMatchFlow);
                    }
                    activeFlowTable.remove(tmpMatchFlow);
                    printActiveFlows();
                    break;

                case STATS_REPLY:
                    OFStatisticsReply statReply = (OFStatisticsReply) msg;
                    ArrayList<OFStatistics> statList = (ArrayList<OFStatistics>) statReply.getStatistics();
```

```
                logger.debug("Received Stat Reply " + statList.size());
                if (statList != null && statList.size() > 0)
                {
                    OFFlowStatisticsReply fstatReply = (OFFlowStatisticsReply) statList.get(0);
                    //duration = fstatReply.getDurationSeconds() +
(fstatReply.getDurationNanoseconds() / 1e9);
                    //duration = matchedFlow.getScheduleTimeout() / 1000.0;
                    duration = (double) fstatReply.getDurationSeconds() + (double)
fstatReply.getDurationNanoseconds() / 1e9
                            - matchedFlow.getDuration();

                    byteCount = fstatReply.getByteCount() - matchedFlow.getMatchedByteCount();
                    logger.debug("Matched Flow Prev. Byte Count = " +
matchedFlow.getMatchedByteCount());
                    logger.debug("Stat reply, Del-byte = " + byteCount);
                    if (byteCount < MIN_SCHEDULE_BYTE_THRESHOLD)
                    {
                        int oldTimeout = matchedFlow.getScheduleTimeout();
                        int newTimeout = Math.min(matchedFlow.getScheduleTimeout() *
SCHEDULE_TIMEOUT_AMPLIFY_FACTOR,
                                MAX_SCHEDULE_TIMEOUT);
                        matchedFlow.setScheduleTimeout(newTimeout);
                        schedule.updateTimeout(oldTimeout, newTimeout, matchedFlow);
                        if (schedule.getAction(newTimeout) == null)
                        {
                            ScheduledExecutorService ses = threadPool.getScheduledExecutor();
                            SingletonTask action = new SingletonTask(ses, new
PollSwitchWorker(newTimeout, this));
                            action.reschedule(newTimeout, TimeUnit.MILLISECONDS);
                            schedule.addAction(newTimeout, action);
                        }
                    }
                    else if (byteCount > MAX_SCHEDULE_BYTE_THRESHOLD)
                    {
                        int oldTimeout = matchedFlow.getScheduleTimeout();
                        int newTimeout = Math.max(matchedFlow.getScheduleTimeout() /
SCHEDULE_TIMEOUT_DAMPING_FACTOR,
                                MIN_SCHEDULE_TIMEOUT);
                        matchedFlow.setScheduleTimeout(newTimeout);
                        schedule.updateTimeout(oldTimeout, newTimeout, matchedFlow);
                        if (schedule.getAction(newTimeout) == null)
                        {
                            ScheduledExecutorService ses = threadPool.getScheduledExecutor();
                            SingletonTask action = new SingletonTask(ses, new
PollSwitchWorker(newTimeout, this));
                            action.reschedule(newTimeout, TimeUnit.MILLISECONDS);
                            schedule.addAction(newTimeout, action);
                        }
                    }
                    matchedFlow.setMatchedByteCount(fstatReply.getByteCount());
```

```
                matchedFlow.setDuration((double) fstatReply.getDurationSeconds()
                        + (double) fstatReply.getDurationNanoseconds() / 1e9);
                }
                break;
            }

            double utilization = (double) byteCount / duration;
            logger.debug("Instant utilization = " + utilization);

            if(flowStatTable.get(matchedFlow) == null)
            {
                flowStatTable.put(matchedFlow, Collections.synchronizedSortedMap(new
TreeMap<Long, Double>()));
            }
            flowStatTable.get(matchedFlow).put(checkPointTimeStamp, utilization);

            if (switchStatTable.get(new Long(sw.getId())) == null)
            {
                logger.debug("Adding a switch entry for the first time, swId = " + sw.getId());
                LinkStatistics linkStat = new LinkStatistics();
                linkStat.setInputPort(matchedFlow.getInputPort());
                linkStat.addStatData(checkPointTimeStamp, utilization);

                SwitchStatistics swStat = new SwitchStatistics();
                swStat.setSwId(sw.getId());
                swStat.addLinkStat(linkStat);
                switchStatTable.put(sw.getId(), swStat);
            }
            else if (switchStatTable.get(new Long(sw.getId())).linkExists(match.getInputPort()) ==
false)
            {
                logger.debug("Adding entry for port = " + matchedFlow.getInputPort() + " on switch "
+ sw.getId());
                LinkStatistics linkStat = new LinkStatistics();
                linkStat.setInputPort(matchedFlow.getInputPort());
                linkStat.addStatData(checkPointTimeStamp, utilization);
                switchStatTable.get(new Long(sw.getId())).addLinkStat(linkStat);
            }
            else
            {
                SwitchStatistics swStat = switchStatTable.get(new Long(sw.getId()));
                synchronized (swStat.getLinkStatTable())
                {
                    for (Iterator<LinkStatistics> lsIt = swStat.getLinkStatTable().iterator(); lsIt.hasNext();)
                    {
                        LinkStatistics ls = lsIt.next();
                        if (ls.getInputPort() == match.getInputPort())
                        {
                            Set<Long> timestampSet = ls.getStatData().keySet();
                            for (Iterator<Long> it = timestampSet.iterator(); it.hasNext();)
```

```
                    {
                        Long ts = it.next();
                        if (ts.longValue() > matchedFlow.getTimestamp()
                                && ts.longValue() <= checkPointTimeStamp)
                        {
                            //logger.debug("[" + matchedFlow.getTimestamp() + "," + ts.toString() +
"," + checkPointTimeStamp + "]");
                            ls.getStatData().put(ts, utilization + ls.getStatData().get(ts).doubleValue());
                        }
                    }
                    ls.addStatData(checkPointTimeStamp, utilization);
                }
            }
        }
        if (type.equals(OFType.STATS_REPLY))
        {
            matchedFlow.setTimestamp(checkPointTimeStamp);
        }
    }
  }
```