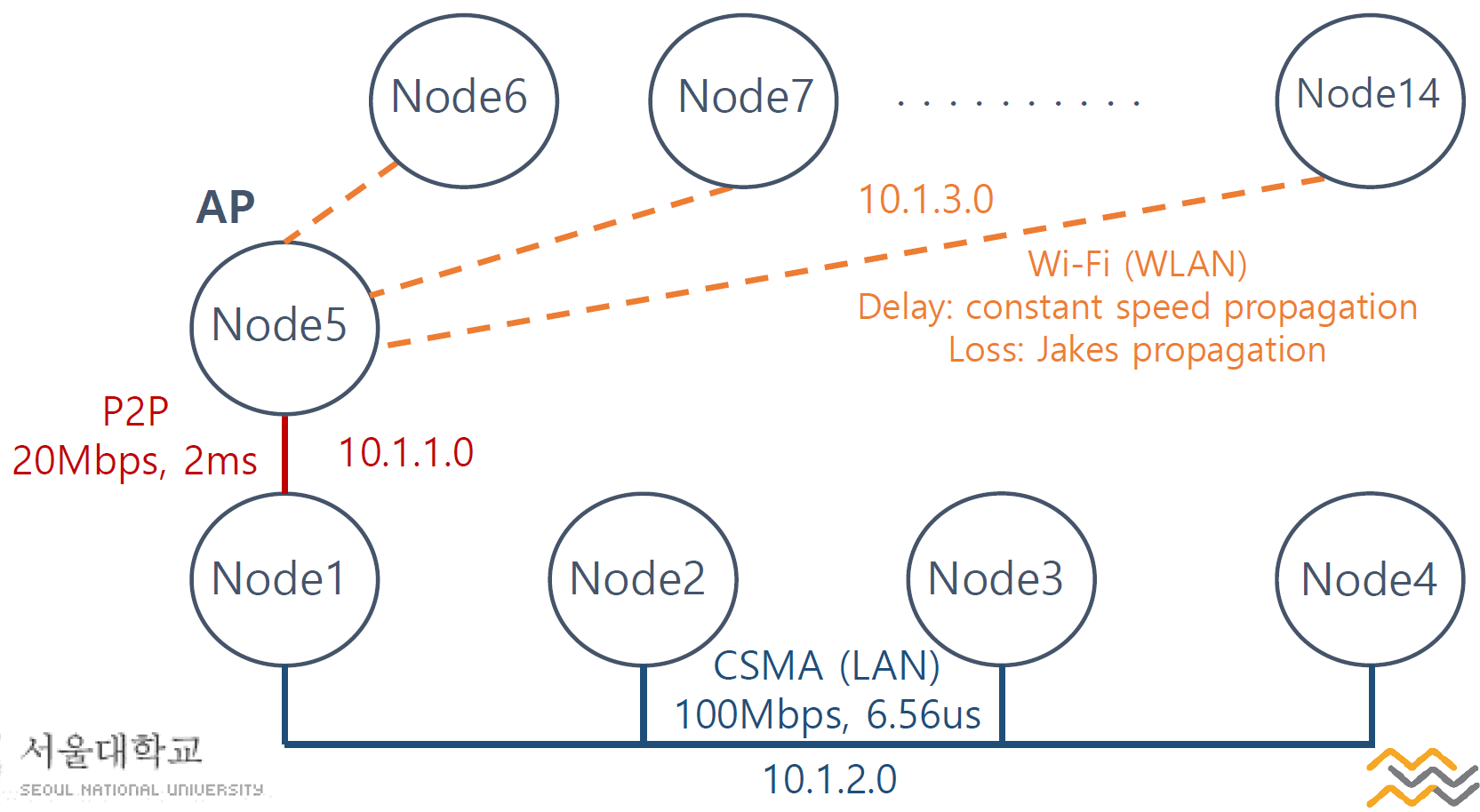
**네트워크 프로토콜 설계 및 실습 과제 7**

전기정보공학부 2016-10769

**1. Objective of experiment**

TCP와 UDP에 대해 두 protocol이 한 channel에 대해 작동할 때의 동작을 확인한다. 또한 TCP의 reliable, in order transmission을 관찰한다. WIFI link를 사용하는 상황에서, UDP를 사용하는 node에 의해 TCP node의 throughput이 어떻게 변하는지 관찰하며 이를 통해 각 protocol 및 channel의 특성을 파악할 수 있다. 이를 ns-3에서 지원하는 tracing system을 통해 확인한다.

**2. Topology & Realization**



**그림1.** 실험에 사용될 topology 개략도

topology는 그림 1과 같이 사용한다. 하나의 AP node가 WIFI를 통해 STA node로부터 data를 받고, 이를 P2P connection을 통해 Node1에 전송하여 CSMA channel로 전달한다. 각 channel의 attribute은 그림1에 표시된 것과 같다. 이 때, node 6~14에서 node 6은 TCP로, 나머지 node는 UDP로 data를 전송한다.

간단한 realization 설명은 다음과 같다. 자세한 사항은 source code의 주석에 설명되어 있다.

Node 6~14는 WIFI station으로, 기본으로는 9개로 설정되어 있으며 이후 UDP data를 전송하는 node를 2, 4, 8개로 조절하기 위해 각각 3, 5, 9개로 설정된다. WIFI channel은 constant speed propagation delay를 가지고, jakes propagation loss model을 상정한다. AP node의 위치는 원점에 고정되어 있으며, STA node들은 rho가 30인 uniform disc position으로 배치되어 RandomWalk2d model을 따라 -30~30의 정사각형의 bound를 가진다. rate control algorithm은 ARF를 사용하였다. 이 때 STA node의 개수는 nWifi 변수로 조절되며, node container wifiStaNodes에 포함되어, 이 container의 첫번째 node는 항상 TCP traffic을 발생시키고 나머지 node는 모두 UDP traffic을 발생시킨다. 이 때, traffic을 발생시키는 application은 in class 3주차에 사용된 application을 이용하였다.

Node 5와 Node1은 각각 AP node와 CSMA를 잇는 역할을 하는데, 이를 위해 Point to Point channel을 사용한다. data rate은 20Mbps, delay는 2ms를 사용한다. Node 1~4는 CSMA를 사용하고, CSMA channel은 100Mbps, 6.56us의 delay를 갖는다. Global routing table을 enable하여 각 IP 주소를 찾아가도록 한다.

tracing system을 이용하여 Rx를 trace하고, callback function은 다음과 같이 구현하였다.

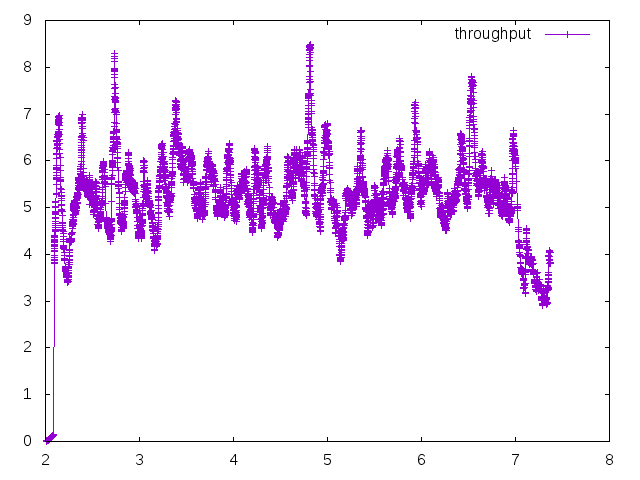
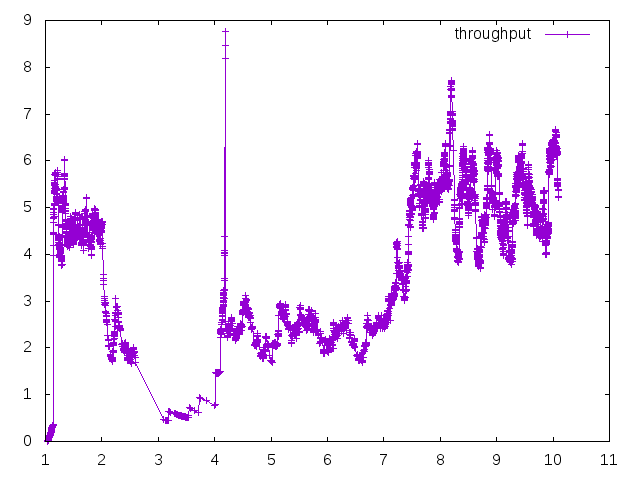
UDP throughput의 경우, CalcThru\_udp function에서 Rx source를 tracing하여 throughput을 계산한다. udp\_data\_in\_window라는 배열은 크기가 100인 int이다. window를 shift하면서 throughput을 계산하기 때문에, packet count가 100보다 작을 경우 지금까지의 모든 data를 더하고, 이를 현재 시간으로 나누어 준다. 100보다 큰 경우, count를 100으로 나눈 나머지를 배열의 index로 사용한다. 이로써 배열을 queue와 같이 사용할 수 있다. 따라서 throughput을 계산하기 위해, 배열의 모든 data 값을 더하고 이를 현재 index의 time 값과 index + 1의 time 값의 차로 나눈다. bit/Mbps로 표현하기 위해 8을 곱하고 106으로 나누어 준다. TCP throughput도 위와 같이 구현되어 있다.

RTS failed의 경우, MacTxRtsFailed trace source를 이용한다. callback function이 불릴 때마다 count를 1씩 증가시킨 후 마지막에 print한다.

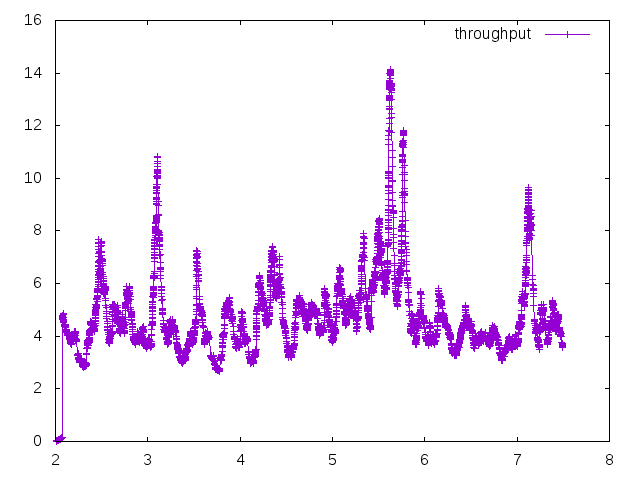
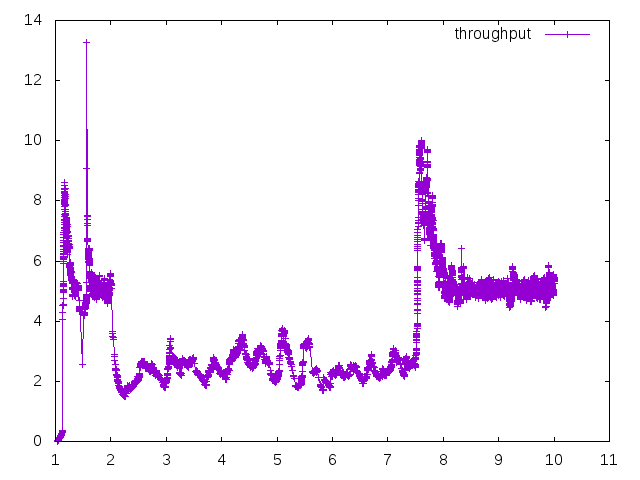
**3. Result & Discussion**

1) Examine the total throughput at node3, node4

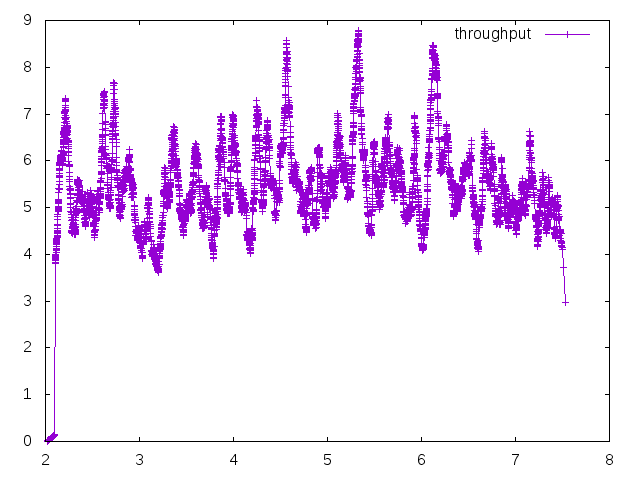
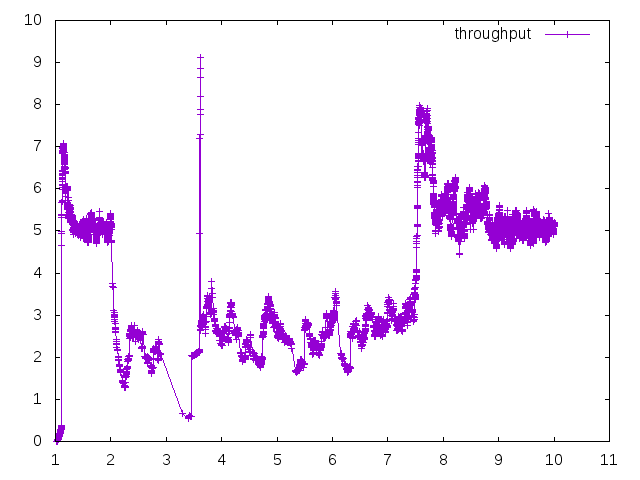
아래의 그래프는 각각의 6개의 case에 대한 결과를 gnuplot을 통해 그린 것이다. 왼쪽은 TCP traffic에 대한 throughput을, 오른쪽은 UDP traffic에 대한 throughput을 나타낸 것이다. x-axis는 second이고, y-axis는 throughput(Mbps)이다.



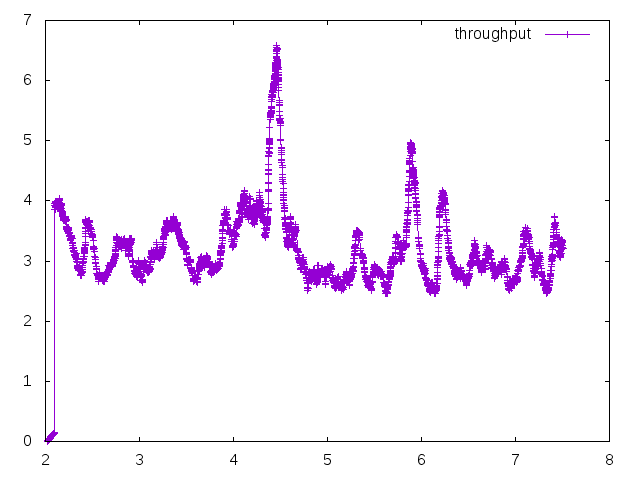
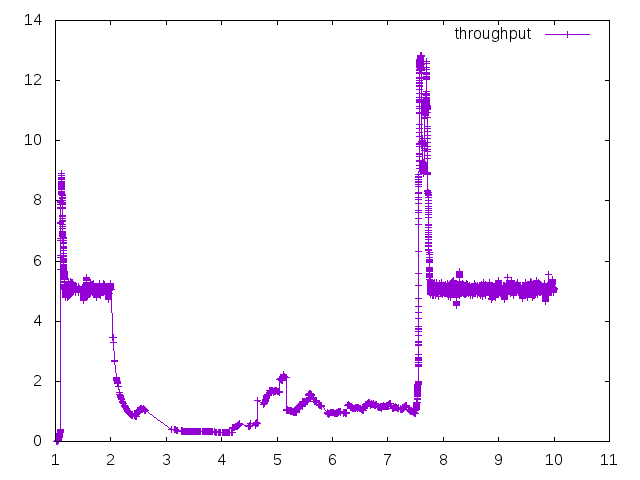
**그림2.** UDP traffic node=2. Threshold=0Mbps.



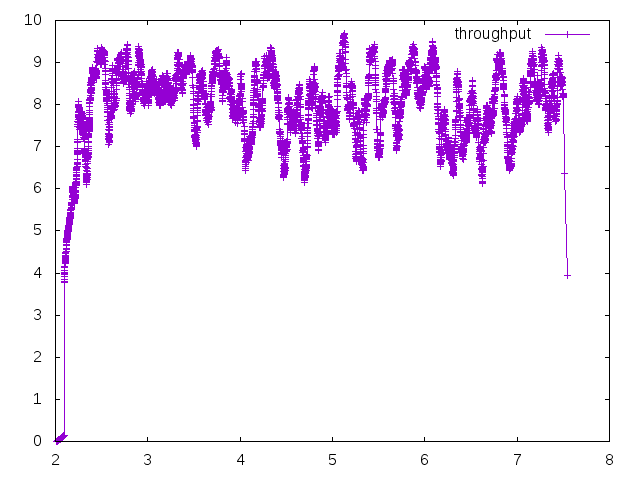
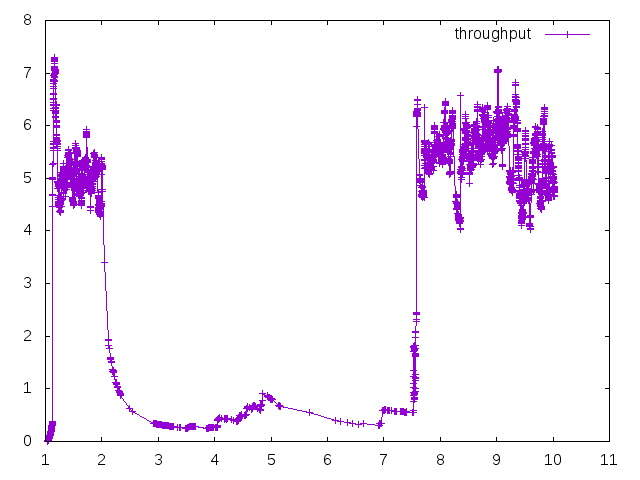
**그림3.** UDP traffic node=2. Threshold=2000Mbps.



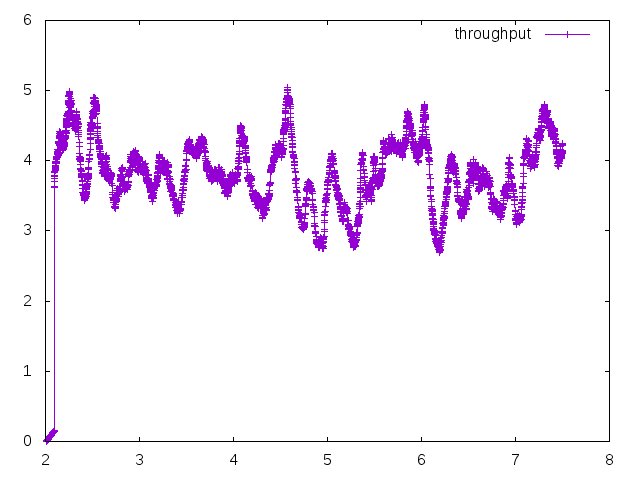
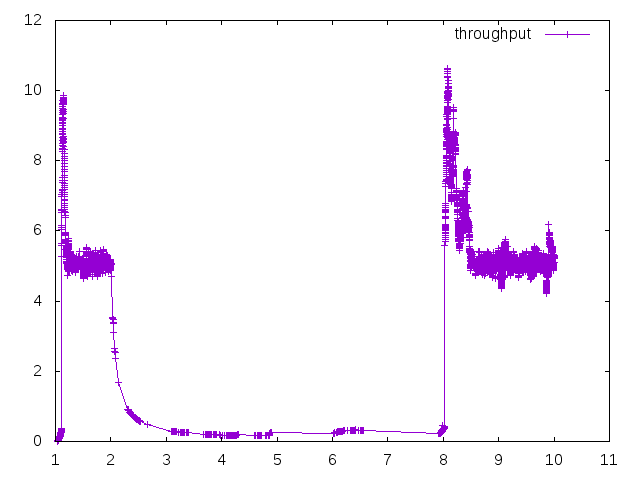
**그림4.** UDP traffic node=4. Threshold=0Mbps.



**그림5.** UDP traffic node=4. Threshold=2000Mbps.



**그림6.** UDP traffic node=8. Threshold=0Mbps.



**그림7.** UDP traffic node=8. Threshold=2000Mbps.

그림 2-7은 6가지 case에 대해 각각 TCP traffic, UDP traffic을 gnuplot을 통해 그린 것이다. Y-axis는 최근 100개의 packet에 대한 average throughput [Mbps]이고, X-axis는 second이다.

먼저 TCP를 살펴보면, TCP traffic은 2)에서의 topology에서 node 6에 의해서만 발생된다. 전반적으로 graph가 비슷하게 나타났는데, 2초까지는 5Mbps주변에서 throughput이 oscillating하는 것을 관찰할 수 있다. 이는 평균적으로 보았을 때, application에서 전송하는 data rate와 비슷하다. oscillation이 발생하는 이유를 분석하면 다음과 같다. 일단 WIFI의 delay, loss model과 Station nodes의 random walk에 의한 것을 생각할 수 있다. WIFI channel에서 delay가 발생하고 packet loss로 인한 retransmission이 발생함으로 인해 Rx throughput의 distribution이 나타나는 것이다. Rx는 TCP sink node에서만 측정되기 때문에 WIFI channel에서의 traffic 변화는 보이지 않는다. 또한, 각각의 node에서 bound를 지정한 random walk가 존재하기 때문에, 각 node로부터의 data loss rate이 달라질 것이다. 또한 TCP의 congestion, flow control에 의한 것을 생각할 수 있다. TCP의 경우 receiver의 window size에 따라 flow를 조절하고, channel에서의 congestion에 따라 send window size를 조절한다. 따라서 이에 따른 throughput도 변화하였을 것이다. out of order로 packet이 완성되지 않았을 때는 모든 transmission이 완료된 후 packet을 application layer로 전달하기 때문에, 이로 인해서도 oscillation이 발생하였을 것이다.

또한 그림 2, 3, 4를 보면 throughput에서 순간적으로 피크 값이 발생한다. 이는 TCP의 send buffer에 의한 것으로 추정된다. application은 시작하자 마자 packet을 전송하기 시작한다. 그러나, TCP는 3-way handshake 과정을 통해 connection을 형성한 뒤에 data를 전송하므로 connection이 establish되기 전에는 packet을 전송할 수 없다. 이로 인해서 buffer에 packet이 쌓이게되고, 이로 인해 peak가 나타난다고 볼 수 있다. 실제로 SYN으로 시작하여 SYN/ACK, ACK까지의 시간은 대략 0.3초정도로 길게 나타났다. 또한, ARF에 의해서 channel의 adaption rate이 커지는 동안에 TCP의 byte stream이 out of order로 전송되어 buffering을 가중시킨 것으로 보인다. 이로 인한 packet의 buffering으로 인해 순간적으로 buffer에서 전송되는 packet양이 증가하는 현상이 관찰되었다고 할 수 있다. 이는 throughput을 Rx side에서 관찰하였기 때문에, application으로 packet이 전달되기 까지의 channel 특성, TCP protocol에 의한 buffering 등이 반영되어 확인되는 현상일 것이다.

TCP의 throughput은 2-7초 사이에 급격하게 감소하였다가 다시 증가하는 모습을 볼 수 있는데, 이는 UDP traffic에 의한 것이다. Point to Point channel과 CSMA에서의 channel rate은 20, 100Mbps로, 2-7초에서의 총 throughput보다 충분히 높게 나타난다. 따라서 이 구간에서의 TCP throughput 감소는 P2P나 CSMA channel의 collision에 의한 것이 아니라, ARF algorithm을 사용하는 WIFI channel이나 TCP protocol 자체에 의한 것이라고 생각된다.

TCP는 channel에서의 collision이 자주 발생한다고 생각되면 congestion control에 의해 sender side의 window를 줄이게 된다. 이는 timeout이나 3-duplicated ACK으로 확인하는데, packet의 loss가 많이 발생하면 congestion avoid mode로 진입하여 transmit하는 packet의 개수를 줄이게 된다. 따라서, UDP traffic에 의해 WIFI channel에서 packet loss가 많이 발생하거나 timeout이 빈번하게 발생하게 되고, 이로 인해 TCP에서의 sending window size가 커지지 못해 이전과 다르게 throughput이 감소하게 되는 것이다.

또한 WIFI channel의 특성에 기인한 것으로 생각할 수도 있는데, ARF algorithm의 경우 collision에 대해 data rate을 줄이는 대처를 취한다. 따라서 이로 인해 전체적인 throughput이 낮아지게 된다. 또한, threshold가 높아짐에 따라 2-7초에서 TCP의 throughput이 급격하게 낮아지는 것을 확인할 수 있는데, 이는 CTS/RTS에 대한 threshold가 증가하여 collision이 더 자주 발생하게 되고, 이로 인해 위에서 설명한 TCP protocol에서의 transmit rate 감소에 의한 것이다.

UDP를 살펴보면, 전체적으로 비슷한 분포를 가진다. UDP traffic의 경우 nWifi가 증가함에 따라 throughput이 조금씩 낮아지는 것을 확인할 수 있다. 또한 UDP throughput에서도 peak가 발생하는데, 이는 TCP에서와 비슷한 buffering에 의한 것으로 추정된다. UDP에서도 packet이 완성되어야 application layer에서 관찰할 수 있기 때문이다.

UDP의 경우 전체적으로 threshold가 낮을 때 높은 throughput을 나타낸다. 이는 nWifi(WIFI station node)의 수가 늘어날수록 확연한 차이를 보이며 관찰된다. node의 개수가 늘어날수록 WIFI channel에서의 collision이 자주 발생하게 되고, 이경우 처음부터 CTS/RTS를 사용하여 collision을 피해 retransmission을 줄이는 것이 throughput을 높이는 데 도움이 되는 것이다.

각각의 TCP sink, UDP sink node에서의 total throughput은 다음과 같다.

|  |  |  |
| --- | --- | --- |
|  | threshold(bytes) | |
| Station node | 0 | 2000 |
| 2 | 3.16672 | 3.14976 |
| 4 | 2.95648 | 2.35712 |
| 8 | 2.08736 | 1.75808 |

**표1.** 각 case별 나타난 total TCP throughput. (단위: Mbps)

|  |  |  |
| --- | --- | --- |
|  | threshold(bytes) | |
| Station node | 0 | 2000 |
| 2 | 4.144 | 3.3392 |
| 4 | 5.8624 | 3.45792 |
| 8 | 8.67328 | 3.8192 |

**표2.** 각 case별 나타난 total UDP throughput. (단위: Mbps)

표 1, 2는 각각 station node의 개수와 threshold를 조절하며 측정한 6개의 case에 대한 TCP, UDP total throughput이다. 이는 전체 Rx packet의 크기를 더한 뒤 Tx time으로 나누어 얻을 수 있었다.

TCP의 경우 station node가 증가함에 따라 throughput이 감소하는 것을 확인할 수 있었다. 또한 그 감소폭은 RTS/CTS threshold가 높을 때 더 크게 나타났다. 앞에서 설명했듯, 이는 WIFI channel에서의 collision에 의한 것으로 추정된다. 이에 따라 TCP protocol에 따른 sender side window 감소, ARF algorithm에 의한 data rate 감소 등으로 인해 throughput이 낮아진 것이다. threshold가 높으면 CTS/RTS가 사용되지 않아 자주 collision이 발생하고 이것이 TCP에서의 collision avoidance를 가중시킨 것이다.

UDP의 경우 station node가 증가함에 따라 total throughput은 증가하였다. 이는 channel에서 UDP가 차지하는 비율이 TCP보다 높기 때문이다. 또한 RTS/CTS가 사용되었을 때 더 크게 throughput이 증가하였는데, 이는 collision이 덜 일어남에 따른 것으로 확인된다. TCP에서 window size를 줄이고 이에 따라 UDP입장에서는 TCP traffic이 줄어들어 channel에서의 collision이 덜 발생하여 throughput이 증가하는 것이다.

2) Trace the total number of RTS packets failed

|  |  |  |
| --- | --- | --- |
|  | threshold(bytes) | |
| Station node | 0 | 2000 |
| 2 | 2306 | 0 |
| 4 | 4017 | 0 |
| 8 | 5199 | 0 |

**표3.** RTS packets failed를 측정한 결과 (단위: 개)

표3은 각 case에서의 RTS packet failed를 측정한 결과이다. threshold가 2000인 경우 RTS/CTS가 사용되지 않았기 때문에 failed packet이 모두 0개로 측정되었다.

station node의 개수가 늘어남에 따라 failed packet의 개수가 증가함을 알 수 있다. 이는 역시 collision에 의한 것으로, station node가 늘어나면서 WIFI channel이 혼잡해지고, 이로 인한 collision을 RTS/CTS로 해결하는데 이 때 RTS끼리도 충돌이 발생한다. RTS는 random하게 전송되지만 station node의 개수가 늘어남에 따라 RTS의 collision 확률이 높아지는 것이다. RTS 전송 timing이 well distributed되어 있다면 traffic과 RTS failed packet의 개수는 선형관계를 가질 것이지만, 이를 확인하기에는 표본의 수가 적었다.

**4. References**

- Kurose, J. F. and K. W. Ross (2012). Computer Networking: A Top-Down Approach (6th Edition).

- ns3 network 3주차 강의 자료.

- ns-3 Doxygen. <https://www.nsnam.org/doxygen/>