BTBB Pandas data analysis

- this example utilizes the btbb scapy libraries along with python pandas to demonstrate how btbb data can be organized and visualized in python
- I am still fairly new to pandas, so some of this could use some restructure

library imports:

In [1]:
```
from scapy.all import *
from btbb import *
from pandas import *
import datetime
```

```
WARNING: No route found for IPv6 destination :: (no default route?)
WARNING:scapy.runtime:No route found for IPv6 destination :: (no default
route?)
```

- open up our btbb pcap file and read in all of our packets
- I am using my stream reader here, but the scapy pcap reader would work too
- note: this could also be done on a live streaming pcap file

In [2]:
```
filename = "../../data/pandas_demo.pcapbtbb"
btbb_stream = BtbbPcapStreamer(filename)
```

In [3]:
```
pcap_pkts = btbb_stream.read_all()
```

- complicated looking but simple loop to create a list of dicts for easy dataframe creation

In [4]:
```
data_list = []
for pcap in pcap_pkts:
    tmp = {}
    pkt = Ether(pcap[0])
    timestamp = float(pcap[1][0])
    addr_list = pkt.src.split(':')
    tmp['time'] = datetime.datetime.fromtimestamp(timestamp)
    tmp['nap'] = ':'.join(addr_list[:2])
    tmp['uap'] = addr_list[2]
    tmp['lap'] = ':'.join(addr_list[3:])
    tmp['type'] = btbb_packet_type[pkt['packet'].type] if pkt.haslayer('Bt
    tmp['to_master'] =  (bin(pkt['meta'].CLK)[2:].zfill(32)[7] == '0') if
    tmp['payload'] = True if pkt.haslayer('BtbbPayload') else NaN
    tmp['name'] = NaN
    data_list.append(tmp)
df = DataFrame(data_list, columns=['nap', 'uap', 'lap', 'type', 'payload',
```

- for those unfamiliar with pandas, here is what a dataframe looks like
- note: packet types marked NaN are empty id packets

In [5]:
```
df.head()
```

Out[5]:

	nap	uap	lap	type	payload	to_master	name	time
0	00:00	00	ed:1d:9c	NaN	NaN	NaN	NaN	2012-06-13 21:06:11
1	00:00	00	e0:55:55	NaN	NaN	NaN	NaN	2012-06-13 21:06:11
2	00:00	00	ed:1d:9c	NaN	NaN	NaN	NaN	2012-06-13 21:06:11
3	00:00	36	ed:1d:9c	POLL	NaN	True	NaN	2012-06-13 21:06:11
4	00:00	00	e0:55:55	NaN	NaN	NaN	NaN	2012-06-13 21:06:11

- lets check out our min and max timestamps in the data

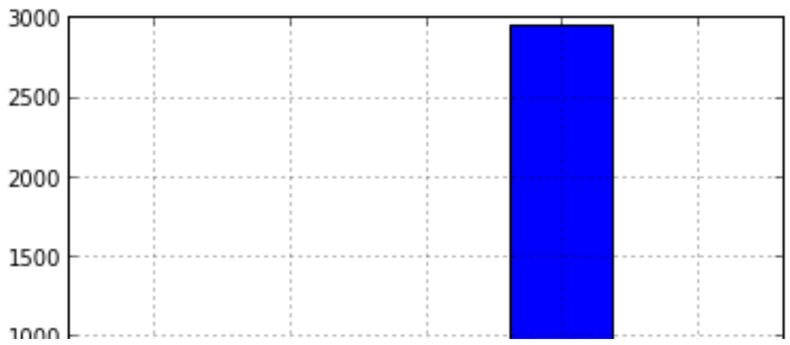
In [6]:
```
print df['time'].max()
print df['time'].min()
```

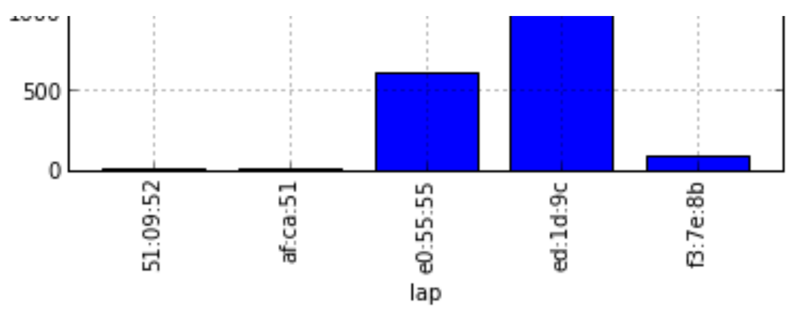
```
2012-06-13 21:11:08
2012-06-13 21:06:11
```

graph count of all clients seen

- toss out clients only seen once
- include id packets

In [7]:
```
seen = df.groupby('lap')
seen = seen.size()
ax = seen[seen > 1].plot(kind='bar')
```

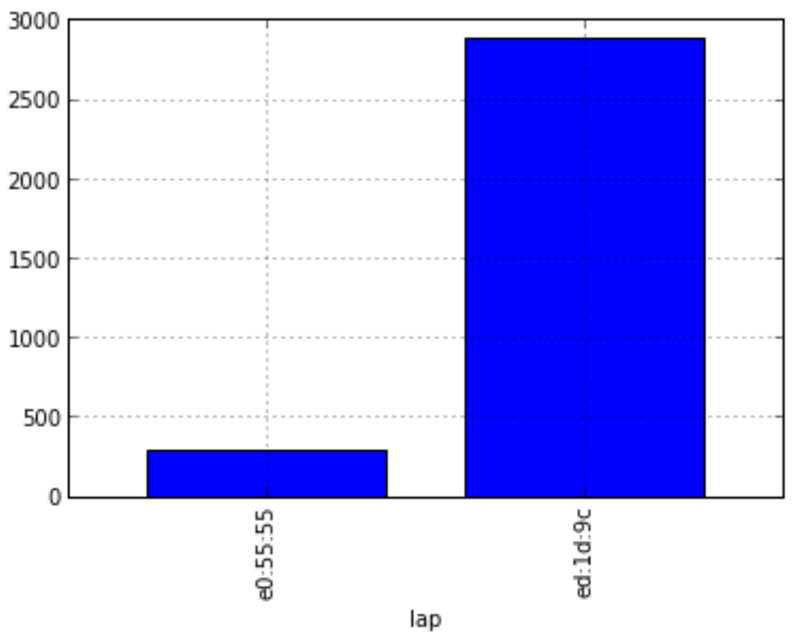




graph no id packets sent per client

- this excludes id packets
- note: more packets were likely sent, but this is all we see on one channel

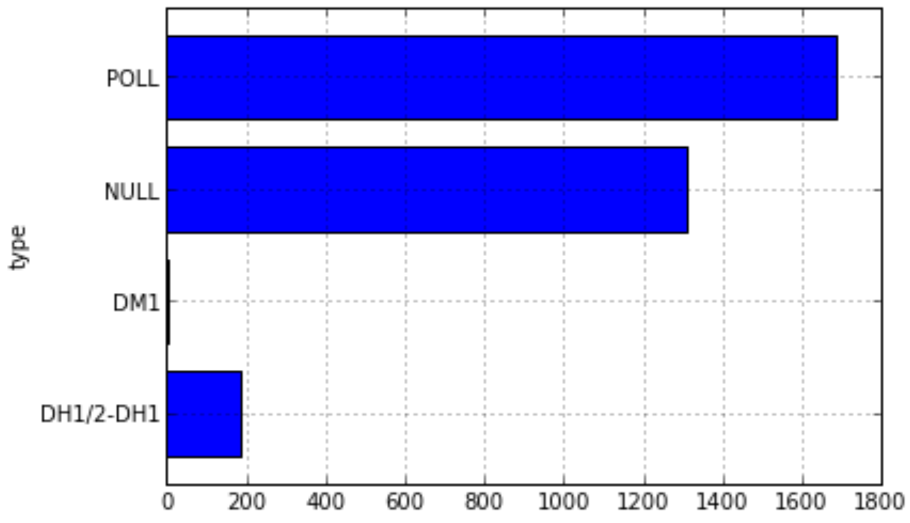
```
In [8]: pkt_count = df[df['type'].notnull()]
        pkt_count = pkt_count.groupby('lap')
        pkt_count = pkt_count.size()
        ax = pkt_count.plot(kind='bar')
```



graph breakdown of non id packets seen

- this excludes id packets
- this is accross all clients seen

```
In [9]: type_count = df[df['type'].notnull()]
        type_count = type_count.groupby('type')
        type_count = type_count.size()
        ax = type_count.plot(kind='barh')
```

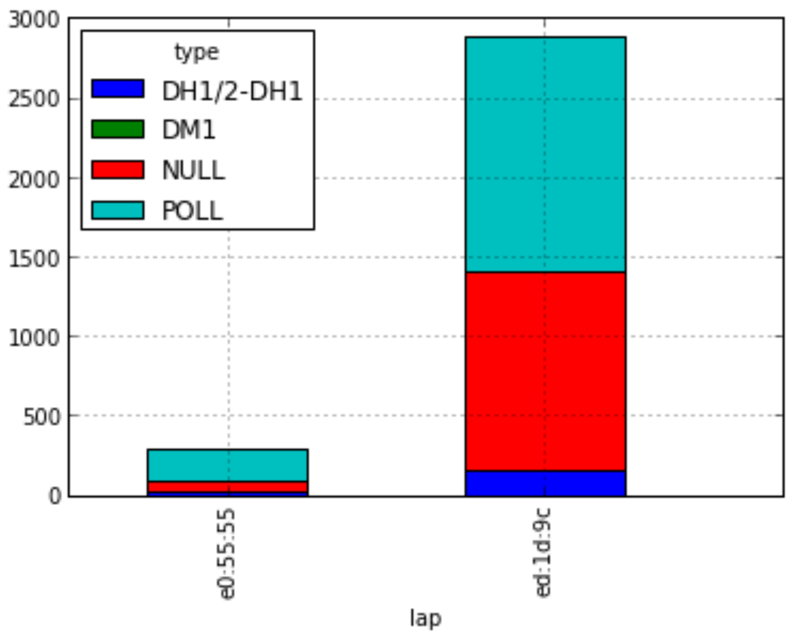


graph non id packets seen per client

- this excludes id packets

In [10]:
```
lap_type_count = df[df['type'].notnull()]
lap_type_count = lap_type_count.groupby(['lap','type'])
lap_type_count = lap_type_count.size()
lap_type_count = lap_type_count.unstack(1)
lap_type_count.plot(kind='bar', stacked=True)
```

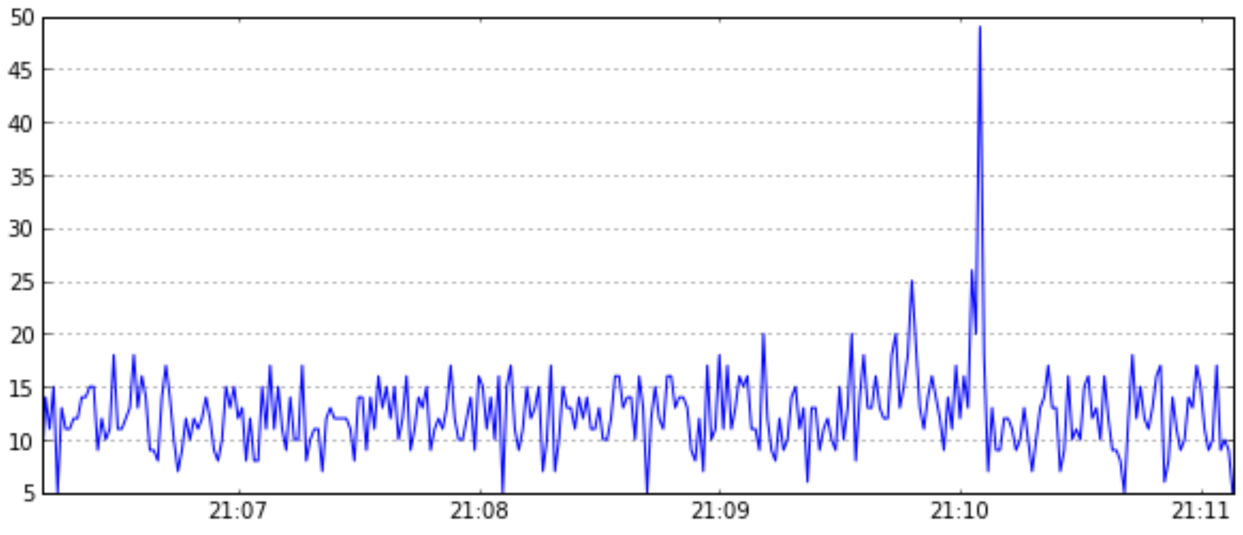
Out[10]:
```
<matplotlib.axes.AxesSubplot at 0x10ba47210>
```



All btbb packets seen over time:

- includes id packets
- taken at a one second frequency

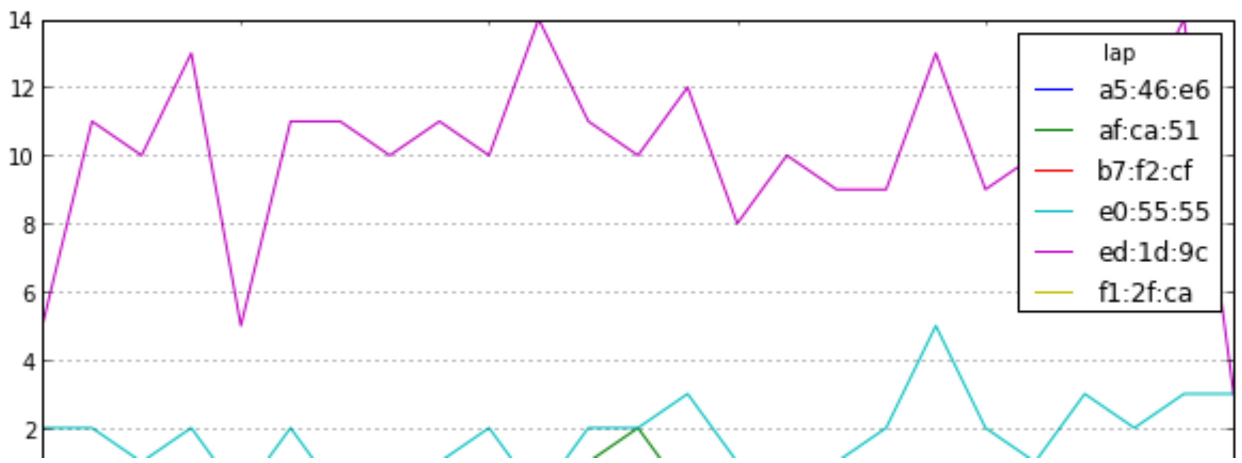
In [11]:
```
ts_seen = df.groupby(['time'])
ts_seen = ts_seen.size()
ts_seen = ts_seen.asfreq('S')
ax = ts_seen.plot(kind='line')
ax.figure.set_figwidth(10)
```

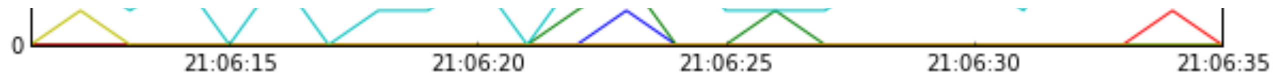


packets seen over time per client:

- sometimes this looks nicer with subplots

In [12]:
```
ts_seen = df[:300].groupby(['time', 'lap'])
ts_seen = ts_seen.size()
ts_seen = ts_seen.unstack(1)
ts_seen = ts_seen.asfreq('S')
ts_seen = ts_seen.fillna(value=0)
ax = ts_seen.plot()
ax.figure.set_figwidth(10)
```

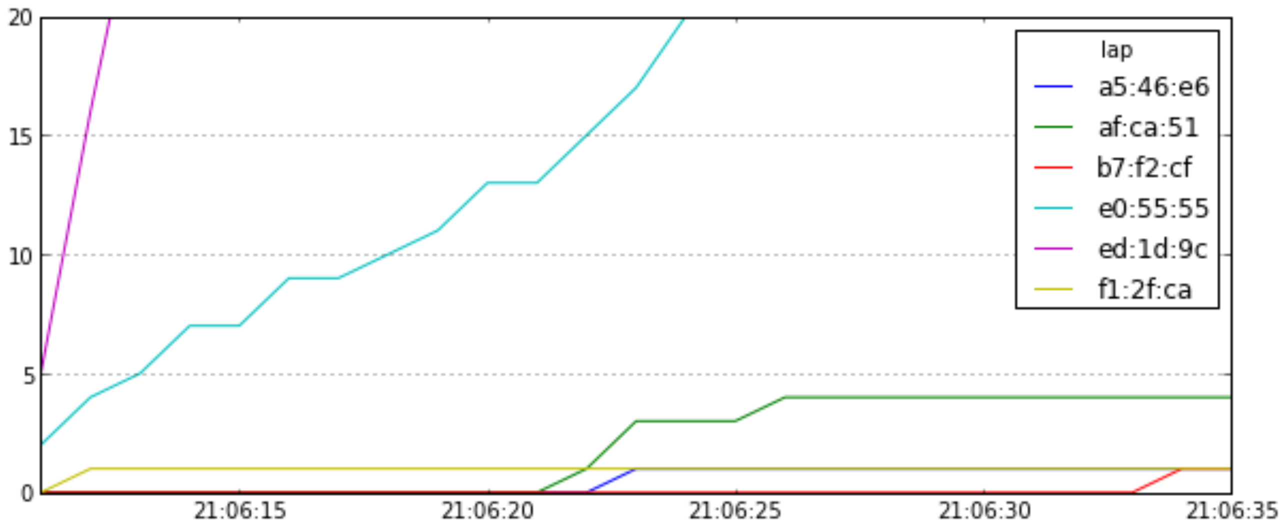




total packet growth over time per client

- similar to the above but shows cumulative sum over time

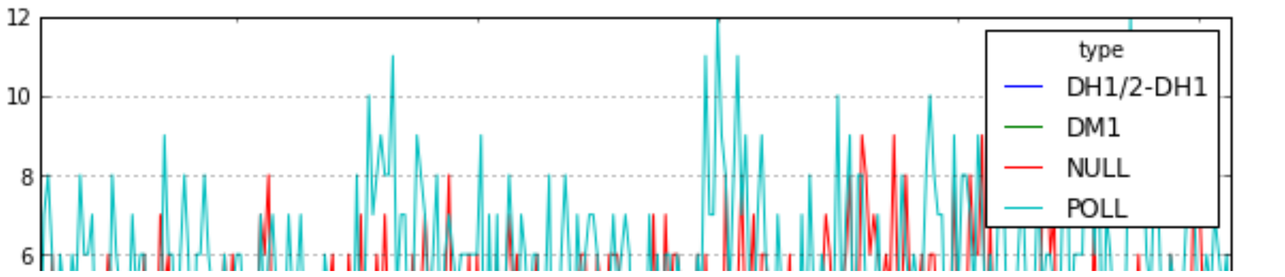
In [13]:
```
ax = ts_seen.cumsum().plot(ylim=(0,20))
ax.figure.set_figwidth(10)
```

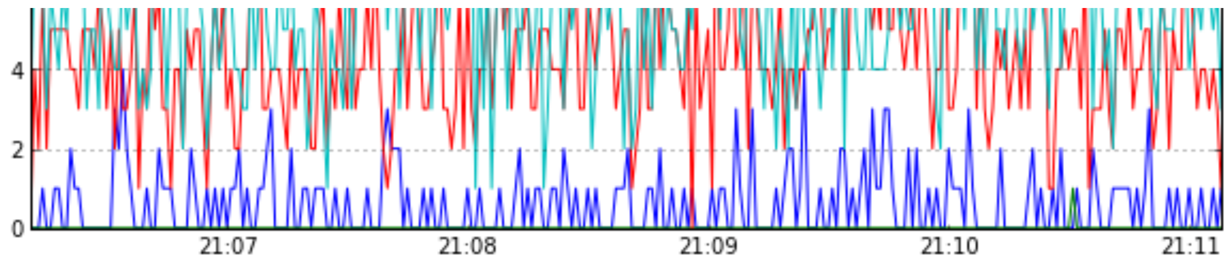


Non id packets seen over time:

- noisy chart
- helpful to use wider graph size, smaller data sets, etc

In [14]:
```
ts_type = df[df['type'].notnull()]
ts_type = ts_type.groupby(['time','type'])
ts_type = ts_type.size()
ts_type = ts_type.unstack(1)
ts_type = ts_type.asfreq('S')
ts_type = ts_type.fillna(value=0)
ax=ts_type.plot()
ax.figure.set_figwidth(10)
```

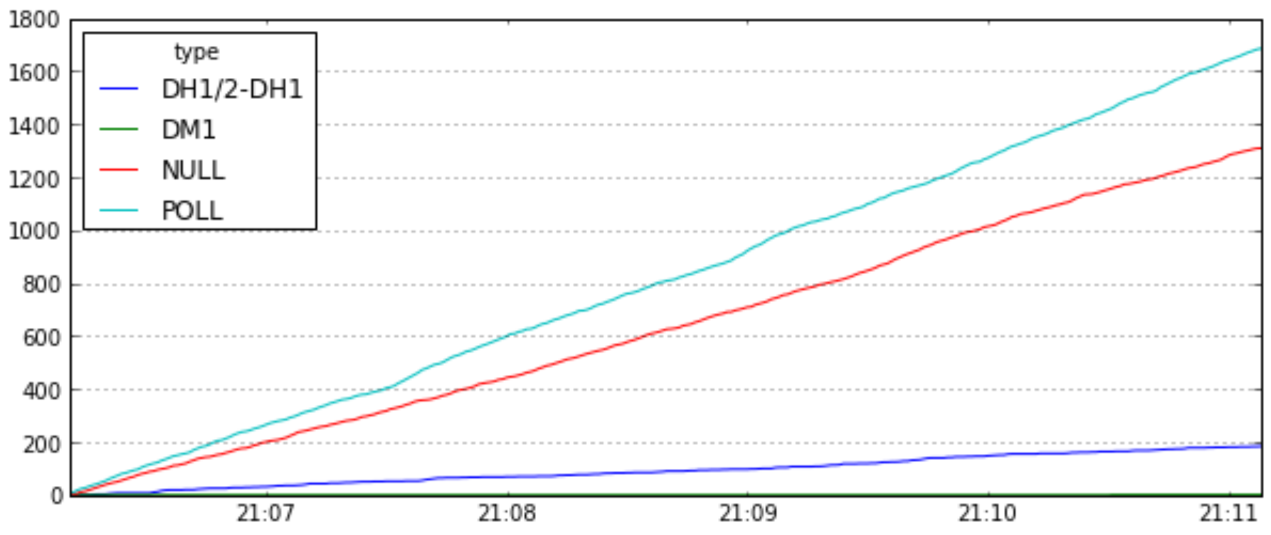




packet type growth over time:

- excludes id packets
- derived from the same data above but much easier to read

In [15]:
```
ax = ts_type.cumsum().plot()
ax.figure.set_figwidth(10)
```



Individual client packets over time:

- list laps seen by clients who have sent non id packets

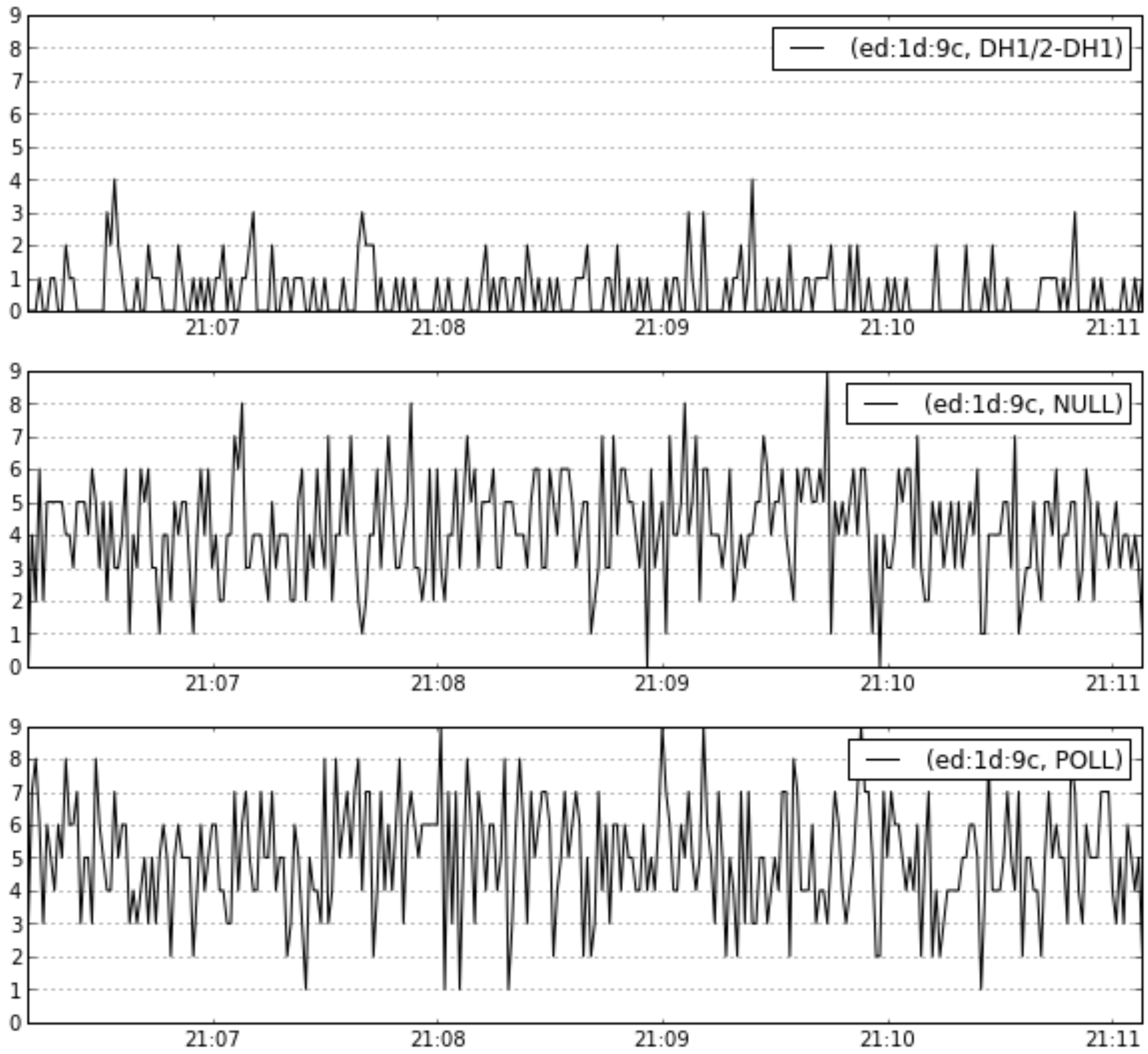
In [16]:
```
laps = df[df['type'].notnull()]
laps = laps.groupby('lap')
sample_lap = laps.indices.keys()[0]
laps.indices.keys()
```

Out[16]:
```
['ed:1d:9c', 'e0:55:55']
```

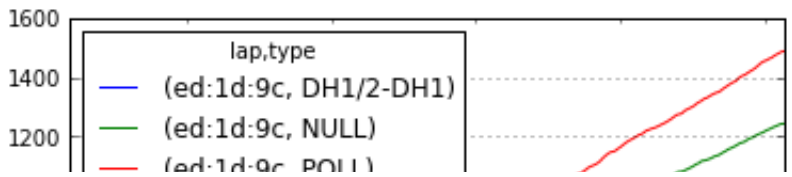
graph packet times seen over time

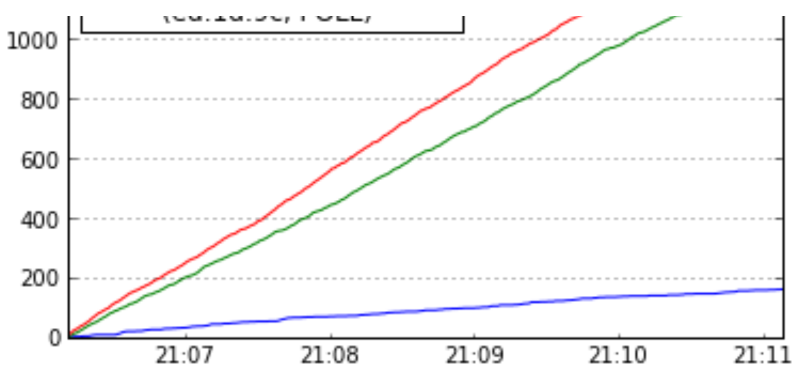
- similar to the cumulative graphs above
- excludes id packets

In [17]:
```
sample_lap_packets = df[df['lap'] == sample_lap]
ts_lap_type = sample_lap_packets[sample_lap_packets['type'].notnull()]
ts_lap_type = ts_lap_type.groupby(['time','lap','type'])
ts_lap_type = ts_lap_type.size()
ts_lap_type = ts_lap_type.unstack(1)
ts_lap_type = ts_lap_type.unstack(1)
ts_lap_type = ts_lap_type.asfreq('S')
ts_lap_type = ts_lap_type.fillna(value=0)
ax = ts_lap_type.plot(subplots=True, figsize=(10,10), sharey=True)
```



In [18]:
```
ax = ts_lap_type.cumsum().plot()
```




(cd:1d:5c, FULL)
1000
800
600
400
200
0
21:07
21:08
21:09
21:10
21:11